UMBC

DISSERTATION APPROVAL SHEET

Title of Dissertation: Advanced Semi-supervised Tensor Decomposition Methods for Malware Characterization

Name of Candidate: Maksim Ekin Eren
meren1@umbc.edu
Doctor of Philosophy, 2024

Graduate Program: Computer Science

Dissertation and Abstract Approved:

DocuSigned by:
[illegible]
188991855180449...
Charles Nicholas
nicholas@umbc.edu
Professor
Computer Science
7/22/2024 | 2:25 PM EDT

NOTE: *The Approval Sheet with the original signature must accompany the thesis or dissertation. No terminal punctuation is to be used.

ABSTRACT

Title of dissertation:	Advanced Semi-supervised Tensor Decomposition Methods for Malware Characterization
	Maksim E. Eren, Doctor of Philosophy, 2024
Dissertation directed by:	Professor Charles Nicholas Department of Computer Science and Electrical Engineering

Malware continues to be one of the most dangerous and costly cyber threats to national security. As of last year, over 1.3 billion malware specimens have been documented, prompting the use of data-driven machine learning (ML) techniques for their analysis. However, existing ML approaches face significant barriers that limit their widespread implementation. These challenges include the detection of novel malware, maintaining performance with low quantities of labeled data during training, and classifying malware under class imbalance: a scenario where malware families are unevenly represented in the dataset. This dissertation addresses these shortcomings by introducing three novel semi-supervised ML methods based on tensor decomposition. Our methods are based on dimensionality reduction, hierarchical tensor decomposition, automatic model determination, and feature extraction methods with selective classification or reject-option capability. This "reject-option" capability is a form of self-awareness that allows our models to abstain from making a decision under uncertainty, which in return allows for detection of novel threats. In this dissertation, we describe the foundational concepts underlying our methods and describe the approaches we developed: the Random Forest of Tensors (RFoT), HNMFk Classifier, and MalwareDNA. Additionally, we detail the capabilities of our methods to utilize High Performance Computing (HPC), multi-processing, and Graphical Processing Units (GPUs) for accelerated computation. We showcase our experiments with all three methods where we demonstrate stable task performance under extreme class imbalance, low-quantity of labeled data, and extreme quantities of malware families. We also showcase results when simultaneously classifying benign-ware and malware, classifying malware families, and detecting novel malware families. Our results are compared against state-of-the-art semi-supervised and supervised ML baselines on two datasets. We showcase how our method surpasses the performance of our baselines with a trade-off in increased abstention or reject-option rate.

Advanced Semi-supervised Tensor Decomposition Methods for Malware Characterization

by

Maksim E. Eren

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Summer 2024

Dedication

To my beloved grandfather Nevzat Eren and grandmother Gonul Hatay Eren.

"You do not have the luxury to shy away from the fight and fall into despair. If you lose your will to fight, what good will your expertise be? ... Your success (if any) is the product of your passion for your profession." - Nevzat Eren

Acknowledgments

I would like to express my deepest gratitude to Dr. Charles Nicholas, who has been an exceptional mentor, advisor, teacher, and supporter throughout my studies at the University of Maryland, Baltimore County (UMBC). His guidance was pivotal in fostering my development as both a student and a researcher. I am also profoundly grateful to my mentor, Dr. Boian Alexandrov, for the privilege of learning about tensor decomposition and machine learning methods under his mentorship. Dr. Alexandrov continuously encouraged me to enhance my skills, explore new areas, and collaborate on innovative projects, which significantly contributed to my success.

I am thankful to Dr. Roberto Yus and Dr. Cynthia Matuszek for their mentorship and support, which have fostered my growth as a researcher. It has been a great privilege to learn about machine learning and data privacy from them. My appreciation also extends to Dr. Manish Bhattarai and my colleagues at Los Alamos National Laboratory (LANL), including Ryan Barron and Nicholas Solovyev, for their mentorship, friendship, and constant encouragement. Continuously inventing and solving problems together with the supportive work culture at LANL with my peers has been an incredibly rewarding experience. I would also like to extend my thanks to my PhD committee members: Dr. Charles Nicholas, Dr. Boian Alexandrov, Dr. Manish Bhattarai, Dr. Roberto Yus, Dr. Cynthia Matuszek, and Dr. Richard Forno, for their time and support.

I would also like to acknowledge all the students from the UMBC DREAM Lab. Collaborating and brainstorming with some of the brightest and most motivated peers I have encountered has been a truly enriching experience. Additionally, I must express my heartfelt gratitude to my family and friends, whose unwavering support has been my constant source of strength.

Special thanks to the Analytics, Intelligence and Technology Division - Advanced Research in Cyber Systems (A-4) group at LANL for their support of my PhD studies. The research presented in this dissertation has been partially funded by A-4. Additionally, this research has received partial funding from the LANL Laboratory Directed Research and Development (LDRD) grant Rapid Response 20230753CR, managed by the Information Science and Technology Institute (ISTI). Other funding sources include LANL LDRD grants 20230067DR and 20190020DR, NA-22 grant SLIC, NA-22 grant ADAPT, as well as the LANL Institutional Computing Program, supported by the U.S. Department of Energy National Nuclear Security Administration under Contract No. 89233218CNA000001. Special thanks to the

Laboratory for Physical Sciences for supporting the work for Random Forest of Tensors (RFoT). This dissertation has been assigned LA-UR-24-24964.

Table of Contents

List of Tables

List of Figures

Executive Summary

Malware is one of the most dangerous and costly cyber threats to national security and a crucial factor in modern cyber-space. The majority of prior research for malware family classification, over the past two decades, has not sufficiently accounted for core evaluation criteria in their work including learning under class imbalance, ability to identify new malware, and the cost of production-quality labeled data [133, 148]. For example, the majority of ML solutions for malware family classification are unrealistically limited to identifying the top most populous families. This results in reports of excellent performance on evaluation metrics that do not generalize to the real world, limited as they have been to the analysis of *"easy"* malware. At the same time, semi-supervised learning in the malware classification field has not been widely explored despite its potential benefits [148]. Finally, proposed ML models are often designed either for malware/benign-ware classification or malware family classification as separate tasks. With the ever-growing quantity of malware, attacks, and their complexities there is an urgent need to improve existing solutions and their operational architectures to drive the increased adaption of ML-based solutions.

In this dissertation we describe three frameworks that addresses the short-

comings of large-scale malware analysis using advanced tensor decomposition methods. While the first two frameworks described in the dissertation, Random Forest of Tensors (RFoT) and HNMFk Classifier, lay the groundwork for addressing the shortcomings, the third framework, MalwareDNA, enhances their ideas and tackles their limitations. MalwareDNA unifies the capability of malware detection and malware family classification into a single framework, while also addressing the shortcomings of novel malware family identification. In this way, MalwareDNA can classify known malware families and separate them from benign-ware, as well as identify new types of families, *all at the same time.* MalwareDNA, in addition, addresses the other aforementioned major shortcomings in the field of malware analysis with ML. Our method operates well under class-imbalance where it can detect both prominent and rare malware. MalwareDNA is based on hierarchical non-negative matrix factorization (NMF) with automatic model determination (NMFk) [10], with the implementation of selective classification (or the reject-option, abstaining classification) [44, 53, 184].

We have thus far reported state-of-the art results in use of tensor decomposition in the field of cyber and data privacy. Our anomaly detection method based on unsupervised tensor decomposition out-performed sev-

Method	**Venue**	**Type**	**Year**	**Citation**
Cyber Anomaly Detection	IEEE International Conference on Intelligence and Security Informatics	Conference Paper	2020	[57]
Cyber Anomaly Detection	ACM Digital Threats: Research and Practice	Journal Paper	2022	[72]
Tensors for Malware AV Scan	Malware Technical Exchange Meeting	Poster	2022	[29]
SCADA Anomaly Detection	IEEE Military Communications Conference	Conference Paper	2023	[129]
Tensors for Cybersecurity	Malware Technical Exchange Meeting	Poster	2024	[68]
Text Mining	ACM Symposium on Document Engineering	Conference Paper	2021	[77]
Text Mining	ACM Symposium on Document Engineering	Conference Paper	2022	[73]
Text Mining	International Conference on Machine Learning and Applications	Conference Paper	2023	[158]
Text Mining	Conference on Data Analysis	Poster	2023	[67]
Knowladge Graphs & Text Mining	IEEE International Symposium on Digital Forensics and Security	Conference Paper	2024	[27]
HPC	IEEE High Performance Extreme Computing Conference	Conference Paper	2022	[32]
HPC	The Journal of Supercomputing	Journal Paper	2023	[33]
Data Privacy	International Conference on Machine Learning and Applications	Conference Paper	2022	[71]
Tensor Decomposition: CP-APR	GitHub	Software	2021	[61]
RFoT	GitHub	Software	2022	[63]
Tensor Decomposition: CP-ALS	GitHub	Software	2022	[55]
Tensor Decomposition Toolbox	GitHub	Software	2023	[69]
RFoT	Malware Technical Exchange Meeting	Poster	2021	[62]
RFoT	University of Maryland, Baltimore County	M.S. Thesis	2022	[56]
RFoT	Springer Nature	Book Chapter	2024	[70]
HNMFk Classifier	ACM Transactions on Privacy and Security	Journal Paper	2023	[74]
HNMFk Classifier	Los Alamos National Laboratory	News Article	2024	[134]
MalwareDNA	IEEE International conference on Intelligence and Security Informatics	Conference Paper	2023	[65]
MalwareDNA	Los Alamos National Laboratory	Patent	2023	[64]
MalwareDNA	Malware Technical Exchange Meeting	Poster	2023	[66]
MalwareDNA	IEEE International Symposium on Digital Forensics and Security	Conference Paper	2024	[76]

Table 1: This table outlines the various outcomes associated with this dissertation, including conference and journal papers, posters, theses, news articles, and patents for each method. Outcomes directly resulting from this dissertation are highlighted in **bold text**. Additionally, the table includes outcomes from other tensor decomposition-based research efforts led by and contributed to by the author of this dissertation.

eral other available state-of-the-art models (including supervised and semi-supervised) in a diverse set of complex anomaly detection tasks including detection of botnet and spam emails, identifying compromised credit cards and users with stolen credentials, and detecting power-grid anomalies [57, 61, 72, 129]. We have also, using tensor decomposition, introduced the first implementation of a one-shot federated learning method for collaborative filtering [71]. Our work thus far provides evidence for promising application of tensors in the realm of cyber and data privacy space, and specifically malware analysis. Earlier ideas behind MalwareDNA were used with another method named RFoT, which we presented at the 2021 Malware Technical Exchange Meeting (MTEM) [62]. RFoT later formed the basis for Eren's M.S. thesis [56], and then got accepted as a chapter in a Springer Nature book [70]. Portions of the MalwareDNA work was published in the ACM Transactions on Privacy and Security (TOPS), specifically the hierarchical bulk classification part, which we named HNMFk Classifier. This was used to classify the largest number of recorded malware families under realistic extreme class imbalance, where our method outperformed the state-of-the-art architectures [74]. In this work, we have also demonstrated how our solution can work with a low quantity of labeled data (stable performance even as the training-set size was dropped to 5%). This work later led to a news article by Los Alamos National Laboratory (LANL) [134]. Since RFoT and HNMFk Classifier lay the groundwork for developing the ideas of MalwareDNA, this dissertation initially outlines these methods.

We have also presented a paper at the 2023 IEEE international conference on Intelligence and Security Informatics (ISI), that showcased the preliminary results from using MalwareDNA on a small quantity of malware data for novel malware family detection [65], presented a poster on MalwareDNA being applied to a larger set of malware specimens at the 2023 MTEM [66], and presented a paper at the 2024 IEEE International Symposium on Digital Forensics and Security (ISDFS) that showcased MalwareDNA's capability to work under class imbalance [76]. Finally, the novelty of the method and the prospect of its application on other U.S. mission-critical problems has led to a 2023 provisional patent submission by LANL for MalwareDNA [64]. We summarize the direct and indirect outcomes of this PhD dissertation in Table 1. This dissertation compiles all the ideas for our tensor decomposition-based, semi-supervised scheme with reject-option, by integrating the concepts of RFoT, HNMFk Classifier, and finally, MalwareDNA. Furthermore, this dissertation provides a more detailed study of MalwareDNA, including new experiments and novel confidence calculations that define the reject-option.

Chapter 1

Introduction

Malware is a generic term for any unwanted software with a purpose of stealing personal or confidential information, or causing harm to the system when deployed. Recent cyber reports rank malware as one of the most costly and frequent cyber threats [31]. In general, the yearly cost of malware to organizations is reported to be \$2.5 million [31], while the total cost of a single ransomware breach is nearly \$4.62 million [2]. In addition, approximately half a million new malware specimens are reported daily [164]. This rapid increase in the quantity of malware is accompanied with the growing sophistication and threat capabilities, making the problem of defending against malware more challenging [3, 4]. The growing capabilities, sophistication, cost, and the quantity of malware combined with the lack of an experienced malware analyst workforce to respond to the immense number of malware attacks drives the need to utilize automated defense systems based on machine learning (ML) to combat malware [106, 126, 144, 145]. Use of ML for automated malware analysis can enable early detection and reduce response times allowing automation to reduce the cost of a security breach by 80% [2].

The goal of malware family classification is to assign malware family labels to known-malware examples to better understand specimen behavior [148]. Malware authors actively generate new specimens to evade detection and to introduce novel threats, resulting in variations within existing malware families or evolution of new/novel families. One aspect that our work focuses on is classifying existing and novel malware families in a multi-class classification setting with uncertainty quantification, an important task for risk analysis to assess threat severity and develop mitigation strategies for emerging or new threats. Our work also demonstrate capabilities in separating malware from benign-ware, a binary classification task. While ML-based solutions may reduce time and costs for malware detection and recovery, the adoption of such strategies has been slow [95]. We attribute this to real-world complexities pertaining to malware analysis [95, 148], and seek to address these shortcomings in this work.

Recent works often overlook relevant evaluation criteria for real-world applications of malware family classification. These core criteria include assessing the model's ability to identify new or novel malware and to classify malware families in the face of class imbalance [74, 75, 133, 148]. Determining that a given specimen is not a member of a known malware family with certainty is an important malware analysis task. At the same time, popular supervised models trained on known malware families may fail to generalize to new data, resulting in false negatives on novel specimens which may lead to security incidences or missed threats [74, 148]. Similarly, the ML-based models should be able to operate under conditions of class imbalance. In malware analysis, class imbalance refers to a large disparity in data class counts– instances of specific malware families significantly outnumber (prominent malware) instances of other low-count classes (rare malware) in the dataset. The models trained with prominent malware families may fail to generalize and detect rare specimens. However, it is still important to detect the rare specimens as they can also cause security breaches. Finally, while semi-supervised learning can help address these shortcomings, they have not been widely studied for Windows malware [148]. With the growing quantities of malware in the wild, there is an urgent need to develop methods that address these shortcomings, and motivate increased adaption of ML solutions.

In this dissertation, we showcase three semi-supervised methods based on tensor decomposition: Random Forest of Tensors (RFoT), HNMFk Classifier, and MalwareDNA. RFoT leverages ensemble learning, together with tensor decomposition and clustering for classifying malware and benign-ware. HNMFk Classifier leverages hierarchical non-negative matrix factorization (NMF) with automatic model determination (NMFk) [10, 11, 12, 13, 132] to perform bulk classification (i.e. no training and test time prediction procedure). While HNMFk Classifier is not a real-time solution, since it is a bulk classifier, it maintains its performance under extreme class imbalance, extreme number of malware families, and in the presence of novel malware families. In contrast, we use MalwareDNA [75], which also uses hierarchical NMFk, as a real-time solution for classification of both rare and prominent malware families (class imbalance) as well as identification of novel malware families.

Both for HNMFk Classifier and MalwareDNA, the automatic estimation of the number of latent (hidden) signatures with NMFk helps avoid under/over-fitting, which enables data modeling with high specificity and accuracy. Meanwhile, the hierarchical clustering approach, guided with a semi-supervised scheme, in our methods helps us determine whatever hierarchical structure exists among the malware specimens and extract accurate patterns or signatures. We use the signatures and the hierarchical clusters for bulk classification with HNMFk Classifier. Differently, with MalwareDNA, we build an archive of latent signatures (identifiers) of malware families.

These latent signatures are then used for precise real-time downstream classification of malware families, and separating them from benign-ware. MalwareDNA also includes a selective classification method to perform abstaining classification (*reject-option*) using distinct confidence metrics [53]. While MalwareDNA uses confidence metrics, HNMFk Classifier includes a different type of reject-option that is based on a semi-supervised scheme, where the labels in the extracted clusters are used as a reference for determining the abstention. The reject-option lets us calculate the confidence of the model, and gives the model the ability to say "I do not know", and gives the user the ability to select between performance and coverage [184]. Here coverage refers to the amount of non-abstaining classification, or the amount of samples where the model did not say "I do not know" and made a decision instead.

The reject-option capability enables HNMFk Classifier and MalwareDNA to identify novel malware families, and maintain its performance under class imbalance by reducing its coverage rate. Abstaining classification with the reject-option capability to withhold from uncertain predictions allow for establishing a framework ideal for "high-stake detection problems as it improves the model reliability and safety" [44]. There are two types of malware features that can be used in ML systems.

Dynamic malware analysis-based features are collected at run-time and often include system calls, file system events, network, and process activity. In comparison, static-malware-analysis-based features directly come from the contents of an executable binary such as the Portable Executable (PE) header, strings, code, and raw bytes. The static malware features can be collected from a specimen without a need to execute the binary. Dynamic analysis-based features often provide a more detailed picture of the malware behavior and are less prone to possible obfuscations and packing techniques [157]. However, obtaining dynamic features has several challenges. Dynamic features require executing the malware in a resource-expensive isolated sandbox environment that often result in a slow feature collection process. In addition, some malware is capable of detecting the presence of a sandbox and modifying its behavior [149]. Despite the potential shortcomings, static malware features are still an effective way to detect and characterize malware. We refer the reader to [157] for more details on classical malware analysis. In this study, we focus on static-malware-analysis-based features. Specifically, we showcase the capabilities of our methods when classifying Windows Portable Executable (PE) format malware specimens into families and benign-ware using static analysis-based features [16].

Our research has already made significant strides, evidenced by our method's publication in peer-reviewed journals, book chapters, news articles, conference proceedings, a master's thesis, poster presentations, and a provisional patent [56, 60, 62, 64, 65, 66, 70, 76, 134]. In this dissertation, we provide a comprehensive overview of MalwareDNA, including discussions on the precursor methodologies, RFoT and HNMFk Classifier, that paved the way for the development of MalwareDNA. Additionally, we delve deeper into the operational mechanisms of MalwareDNA, presenting new experiments and introducing new methods for estimating model confidence, which is crucial for the reject-option mechanism. To the best of our knowledge, our work is pioneering in integrating real-time malware detection, malware family classification, and the identification of novel malware families into a unified system, leveraging selective classification for enhanced security measures. Further, to the best of our knowledge, our experiments with HNMFk Classifier holds the record for the largest number of malware families used for classification all at once (over 2,800).

1.0.1 Motivation and Problem

The rationale behind the shortcomings of the existing ML methods for malware analysis, which are the motivation behind RFoT, HNMFk Classifier, and MalwareDNA, are:

- ML systems used in malware classification often rely on data-hungry methods requiring large quantities of pre-labeled data for training [7, 152], but obtaining large amounts of labeled data is often challenging. For example, labeling malware requires expert knowledge, making the dataset curation expensive and time-consuming [40, 148]. At the same time, problems pertaining to large-scale malware analysis produce large quantities of raw data. An example of this problem is the quantity of malware found in the wild, where approximately 13.5 million new malware specimens are reported every month [148, 165]. Therefore, in global security applications for malware analysis, ML systems need to be able to work with both limited (intrinsic lack of labeled data) and large datasets (large-scale malware analysis).

- Detection of unseen malware (e.g., novel malware family), is an important capability for developing mitigation strategies for new threats and curating new datasets. Current ML systems often do not possess this ability and often do not generalize well to new data [72, 148]. Thus, these ML systems are making increasingly inaccurate predictions with new data-samples.

- Poor generalization to new data poses challenges for class-imbalance problems where a model yields accurate predictions on the prominent threats while failing to recognize the threats with rare data [40]. Here, class-imbalance problem refers to the situation where certain malware families are heavily represented in the dataset as compared to few other families that have low quantities of data. For example, a malware defense systems that is trained on common/prominent malware may fail to detect rare malware [148]. To ensure better defense, ML systems still need to be able to detect the rare families; however, many ML methods require large quantities of data to learn the patterns. Therefore, they often generalize poorly to the families with low-quantities of data and make mistakes.

- Finally, mission-critical objectives in malware analysis generate large quantities of mixed individual signals that require unmixing and understanding of the origin of individual signals for decision making. An illustrative example for mixed signals is the famous ballroom example, where a source microphone may record sound present in the ballroom, and the recorded sound would include mixed signals from everyone present in the room. For accurate identification of individuals present in the room, an ML model would need to separate the mixed signals, which is also known as *blind-source separation*. The hypothesis here is that malware consists of mixed signals from different sources because malware families change over time, retaining patterns from their evolution. An analogy to malware evolution is a biological phylogenetic tree, where a particular specimen (in our case, a malware family) may retain characteristics or patterns from its ancestors in the tree. Therefore, classifying malware families requires separating and extracting individual patterns, which can provide valuable insights into specific malware characteristics, such as the presence of an encryptor or a backdoor routine. The ability to estimate the number of individual hidden signals is needed here, but to the best of our knowledge, is not yet present in any malware family classification tool.

Here, as our ultimate model, we propose a novel semi-supervised AI framework, named MalwareDNA, that addresses the above shortcomings, and that can predict both "This is a known malware!", and "I do not know what this is!" [54, 184], using both limited or extra-large datasets. In this way, MalwareDNA can classify known malware families and separate them from benign software, as well as identify new types of malware families, all at the same time.

1.0.2 Dissertation Statement and Solution

In this dissertation, we introduce three novel semi-supervised AI frameworks designed for large-scale malware analysis, addressing the limitations of current methods. These frameworks, named RFoT, HNMFk Classifier, and MalwareDNA, progressively build upon each other's capabilities. RFoT and HNMFk Classifier lay the foundational groundwork that leads to the development of MalwareDNA, which we consider as our ultimate model. This model not only addresses the limitations of its predecessors but also introduces significant advancements.

MalwareDNA distinguishes itself by its ability to differentiate malware from benign software, classify known malware families, and detect previously unseen types of malware or novel malware families. The innovative aspects of MalwareDNA include several key components:

- An innovative semi-supervised hierarchical method to construct latent signature archives, which includes the ability to work with limited data, to unmix records and determine the number of original signals. This method leverages the ability of tensor decomposition to extract complex and multi-faceted activity details combined with LANL-patented technique for automatic model determination that enables accurate data modeling [10], which has already allowed us to report state-of-the-art unsupervised cyber anomaly detection results in diverse set of difficult problems [57, 61, 72]. The new semi-supervised hierarchical method here further advances the modeling of data for improved specificity and accuracy of latent signature extraction. These signatures are then used for precise real-time downstream classification and identification of malware families, and benign software.

- An innovative optimization method to perform real-time detection of unseen signatures (or novel malware) by implementation of the reject-option method [184]. Reject-option introduces the self-awareness capability to our model, or an ability to know when it does not know [184]. This way, MalwareDNA can refuse to make a decision when it is uncertain, instead of confidently making a wrong decision in contrast to the available traditional ML methods. For example, if a patient visits a doctor for an illness, and the doctor does not know why the patient is sick, it would be preferable to hear from the doctor that the results were inconclusive rather than getting an inaccurate diagnosis. As Albert Einstein once said, "If knowledge is power, knowing what we do not know is wisdom" [184]. Therefore, abstaining from prediction *enable model knowledge growth by the discovery of novelty* (*unknown rejection*) and *establish model security and safety with the recognition of uncertainty* (*failure rejection*) [184]. Here failure rejection enables our method to perform well under class-imbalance, and maintain its performance even when the low-quantity of labeled data is used in training because the model knows when it does not know, and can reject to make a prediction in favor of keeping the non-rejected predictions accurate [59]. Meanwhile, unknown rejection allows the discovery and learning of new knowledge, more specifically, detection of novel malware families. This makes MalwareDNA a good fit for mission-critical problems, which include large-scale malware analysis, where the cost of erroneous predictions could be higher than the cost of rejections or abstaining predictions [44].

MalwareDNA is be able to:

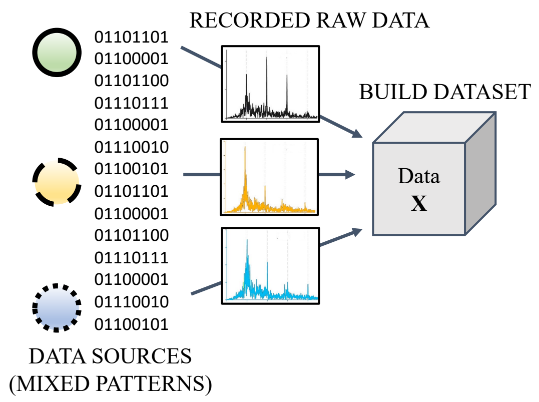
- Accurately extract latent (not directly observable) identifiers/signatures, from both limited (small quantity of labels), or extra-large unlabeled multi-dimensional datasets of malware, and to build archives of these signatures (Figure 1.1-a, Figure 1.1-b, unsupervised).
- Classify in real-time incoming new data malware specimens using these archives (Figure 1.1-c, semi-supervised),
- and finally, recognize in real-time, previously unseen malware families and add them to the archive (Figure 1.1-d, semi-supervised).

The dissertation statement is as follows:

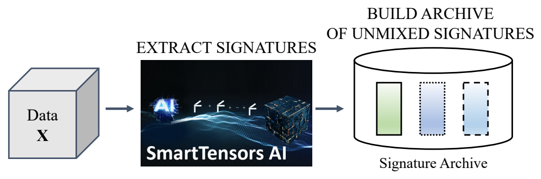
By utilizing powerful tensor decomposition methods, and combining its capabilities with a hierarchical and semi-supervised approach, we can extract accurate latent (hidden) signatures or patterns that characterize malware. These signatures can then be used for downstream classification and detection tasks. In addition, by incorporating reject-option into our system, we add the capability to our method to detect novel malware families, which facilitates knowledge discovery with rejecting to classify unknown samples, making our system safer and better fit for safety-critical applications. With reject-option, our model will avoid making confident wrong decisions, which in return allows maintaining its performance under class-imbalance and under the presence of low-quantities of labeled data, which are major the shortcomings in the field of large-scale malware analysis.

The rest of the dissertation is organized as follows: Chapter 2 provides the necessary background on dimensionality reduction methods, notation, performance evaluation metrics, experiment and baseline configurations, shortcomings in large-scale malware analysis, and the datasets used in the dissertation. Chapter 3 describes the relevant prior research in the field of large-scale malware analysis and dimensionality reduction for malware analysis. Chapter 4 introduces the RFoT method and presents the experiments conducted for

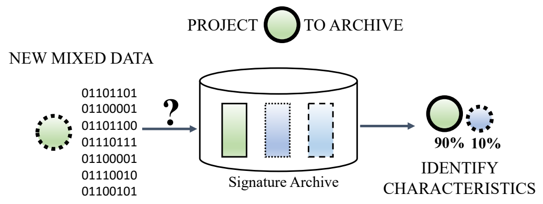
RFoT. Chapter 5 discusses the HNMFk Classifier method, our experiments with this method, and the ablation studies. The MalwareDNA method and its experiments are detailed in Chapter 6. Chapter 7 addresses open-source software considerations and outcomes of the dissertation. Chapter 8 concludes the dissertation, and Chapter 9 outlines future work.



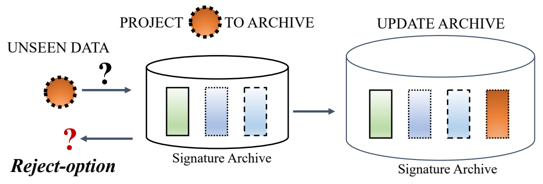
(a)



(b)



(c)



(d)

Figure 1.1: **(a)** Collection of multi-dimensional data. **(b)** Building of latent signature archive. The figure illustrates a blind source separation - unmixed signals stored in the archive. **(c)** Real-time identification of signatures or their combinations, whose components are already in the archive. **(d)** Real-time reject-option that can identify unseen signatures and update the archive.

Chapter 2

Background

In this section we give the background needed for the rest of the dissertation. We first reiterate the relevant challenges associated with large-scale malware analysis in Section 2.0.1 to motivate the methods developed in this dissertation that address these challenges. We follow this section with the notations for tensor decomposition methods in Section 2.0.2. We then summarise tensor factorization in Section 2.0.3, NMF in Section 2.0.4, and NMFk for automatic model determination in Section 2.0.5. Then, we present the intuitive explanation of the ideas behind the hierarchical application of NMFk in Section 2.0.6. Section 2.0.7 describes the datasets used in our experiments and the pre-processing considerations we have taken, while Section 2.0.8 describes our baselines and experiment setups. We close this section with the performance evaluation metrics used in our experiments in Section 2.0.9.

2.0.1 Challenges in Large-Scale Malware Analysis

Malware classification is a challenging task, and the quantity and complexity of malware continues to increase rapidly. This makes ML-based malware classification an important field of study. Raff et al. surveys over 200 research articles on ML-based malware analysis [148]. This survey of the field emphasizes that the standard ML model evaluation technique, where the dataset containing malware families are divided into training and test sets, is flawed when it comes to the malware family classification problem in the real-life case, since previously unseen malware families will continue to appear. To this end, they recommend that the ability to perform abstaining prediction can assist analysts in identifying novel malware. However, prior work has not widely studied this open problem area for malware classification. In our experiments, we evaluate the performance of our solution by including a set of malware families that were not present in the known or training set, as shown in Sections 5.0.2 and 6.0.5.

The second challenge is with regard to the need for separate models for malware classification and malware family classification tasks. Several ML solutions have previously been introduced for distinct tasks of malware detection and malware family classification. The objective of malware detection is to identify a given file as benign or malicious. In contrast to malware detection, malware family classification assumes that any given sample is already known to be malicious, and we want to know to which family it belongs[148]. Existing solutions often use separate ML systems, where one system may be used for detecting malware, and another system is then used to classify the detected malware into a given family. A system that can unify these tasks, such as our proposed solution MalwareDNA, would have operational benefits such as reducing the complexity of maintaining separate systems. The following challenges pertain to other large-scale malware analysis research.

The majority of prior research for malware family classification, over the past two decades, has not sufficiently accounted for core evaluation criteria in their work including learning under class imbalance, ability to identify new malware, and the cost of production-quality labeled data [133, 148]. For example, the majority of ML solutions for malware family classification are unrealistically limited to identifying the top most populous families. This results in reports of excellent performance on evaluation metrics that do not generalize to the real world, limited as they have been to the analysis of *"easy"* malware. However, in production systems where the malware classifiers are used as a critical part of cyber defense systems, the models are exposed to both prominent and rare malware. This is a high-risk scenario where identification of both rare and prominent malware is needed. This is called the class-imbalance problem, and the proposed solutions need to be evaluated with under class-imbalance to report realistic results. An illustrative example of class-imbalance problem is shown in Section 2.0.7 with the EMBER-2018 dataset. The EMBER-2018 dataset is extremely imbalanced, and we showcase how our method handles class imbalance with HNMFk Classifier in Section 5.0.2, and with MalwareDNA in Section 6.0.5.

In the same vein, many malware classifiers are often supervised models that require large quantities of labeled data for training. Since labeling of malware is expensive, the proposed solutions need to be able to yield good malware detection performance even when trained with low quantities of labeled data. In Section 5.0.2 we showcase how HNMFk Classifier handles label scarcity and maintains its performance even with low-quantities of labeled data by compromising the coverage rate. We also demonstrate this capability with RFoT in Section 4.0.5.

At the same time, semi-supervised learning in the malware classification field has not been widely explored despite its potential benefits [148]. Compared to supervised methods, semi-supervised models often yield better generalization to new data. This is especially critical given the amount of new malware reported every day. The high number of new data may result in distribution shift, where a supervised model may begin losing performance as the new samples appear. One approach to defend against the distribution shift is regular re-training of the model; however, this is coupled with the issue of the cost associated with labeling new malware data. Therefore, a model that can maintain good performance via superior generalizability to new data would have significant advantages and cost savings as automated tool for large-scale malware analysis. All methods described in this dissertation (RFoT, HNMFk Classifier, and MalwareDNA) are semi-supervised approaches.

2.0.2 Tensor Decomposition Notation

Tensor decomposition is a powerful data analysis method capable of extracting complex patterns from data in an unsupervised manner. By utilizing tensors, data can be represented in a multi-dimensional space, allowing for the simultaneous exploration of natural relations within and between each dimension. This higher-dimensional and more complex representation facilitates the discovery and interpretation of hidden, multi-perspective information within the data. This section aims to summarize the tensor notations. A summary of the notations utilized in this dissertation can be found in Table 2.1.

Notation	Description
x	Scalar
$\mathbf{x}$	Vector
$\mathbf{X}$	Matrix
$\mathcal{X}$	Tensor
$\mathbf{x}_i$	ith element in the vector
$\mathbf{X}_{i,j}$	Entry located in row i and column j
$\mathbf{X}_{i:}$	ith row
$\mathbf{X}_{:j}$	jth column
$\mathcal{X}^{(i)}$	Superscript (i) used to identify the ith random tensor
$\mathcal{X}_{::j}$	jth slice of a tensor
$\circ$	Outer product

Table 2.1: Summary of the notations used in this dissertation.

Tensors represent a higher-order extension of matrices and enable the representation of multi-dimensional data. A first-order tensor corresponds to a vector, an order-2 tensor is referred to as a matrix, and any structure with dimensions ranging from 3 through D is denoted as a tensor. Specifically, a D-dimensional tensor is termed an order-D tensor, with each dimension also being named the mode of the tensor. For instance, the first dimension of a tensor is also known as the first mode.

To illustrate the construction of a tensor, let's first consider matrices within a lower-dimensional space. For instance, a matrix $\mathbf{X}$ can represent a bag of words extracted from a collection of scientific papers, possessing dimensions *Documents* - *Words* with a shape of $N_{Documents} \times N_{Words}$. In this matrix, an entry $\mathbf{X}_{i,j}$ signifies the number of times word j appears in document i.

Utilizing tensors allows for the creation of a higher-dimensional representation of the data. Taking the example of scientific papers and including additional information such as the publication year for each paper, we can represent this data as an order-3 tensor $\boldsymbol{\mathcal{X}}$ with dimensions *Documents* - *Words* - *Time* and a shape of $N_{Documents} \times N_{Words} \times N_{Time}$. In this tensor, an entry $\boldsymbol{\mathcal{X}}_{i,j,k}$ denotes the frequency of word j appearing in document i at time k (for instance, by year).

We can extend this example from a 3-dimensional tensor to a D-dimensional tensor $\boldsymbol{\mathcal{X}} \in \mathrm{IR}^{\mathrm{N_1 x N_2 x \cdots x N_D}}$, where an entry in the tensor is denoted as $\boldsymbol{\mathcal{X}} i_1, i_2, \cdots, i_D$ and the indexing ranges of each mode are $i_1, i_2, \cdots, i_D \in [0 \leq i_1 < N_1, 0 \leq i_2 < N_2, \cdots, 0 \leq i_D < N_D]$. To facilitate notation, we adopt the *multi-index* notation and use $\mathbf{i}$ to represent the indexing of D modes, such that $\mathbf{i} = i_1, i_2, \cdots, i_D$ and $\boldsymbol{\mathcal{X}}\mathbf{i}$ denotes the tensor entry [43, 92].

Let $nnz(\boldsymbol{\mathcal{X}})$ represent the set of all non-zero entries in the tensor $\boldsymbol{\mathcal{X}}$, and let Ω denote the set of all entries, including the zeros, indicating that we have a sparse tensor when $nnz(\boldsymbol{\mathcal{X}}) < |\Omega|$ [43]. Here, $|\Omega|$ denotes the size of the tensor and is calculated as follows:

$$|\Omega| = \prod_{d=1}^{D} N_d \tag{2.1}$$

We can use the number of non-zeros and the size of the tensor $\boldsymbol{\mathcal{X}}$ to calculate the sparsity of the tensor as follows:

$$\eta = \frac{nnz(\boldsymbol{\mathcal{X}})}{|\Omega|} \tag{2.2}$$

Tensors generated from cyber data often exhibit both extreme sparsity and substantial size. For instance, tensors derived from cyber Netflow data can demonstrate sparsity as low as $\eta = 10^{-8}$ [58]. In the context of Netflow data, the tensor's dimensions might represent the source and destination devices and the timing of network communication events. As shown in Figure 2.1, we depict a tensor with dimensions *User-Source-Destination*, where each binary (0 or 1) entry signifies a *user* engaging in an authentication activity from a *source device* to a *destination device*. Due to the limited communication between most devices within a network, tensors derived from Netflow

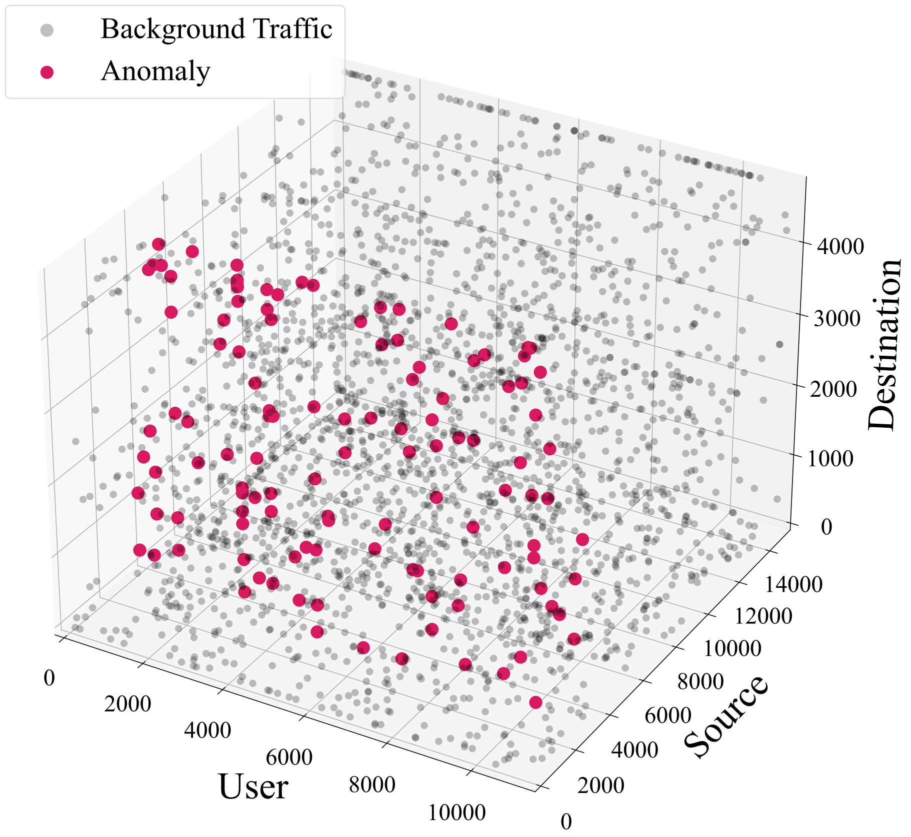


Figure 2.1: Binary tensor with the dimensions *User* - *Source* - *Destination*. The background traffic is shown with gray and anomalies are highlighted in red.

data often display sparsity (as demonstrated in Figure 2.1). Similarly, diverse malware features, such as file size, number of sections, and timestamps, can function as dimensions in the tensor. Each specimen's feature space aligns with a single index along each tensor dimension, akin to the Netflow example, leading to a sparse tensor.

Since tensors can reach large sizes and thus require large amounts of memory, leveraging the sparsity of tensors offers an opportunity to circumvent storing the entire tensor in memory. Instead, the tensor can be stored in a Coordinate (COO) format, which comprises a list of non-zero coordinates and their corresponding non-zero values. In the COO format, each coordinate represents the indexing $\mathbf{i}$, while each non-zero value corresponds to the entry $\mathcal{X}_{\mathbf{i}}$.

2.0.3 Tensor Factorization

The higher-order representation of data using tensors allows for the analysis of latent (hidden) information within the data by considering interactions simultaneously across each dimension, thus facilitating the extraction and discovery of complex and multifaceted details. Two widely used tensor decomposition algorithms are the Tucker decomposition, and the CANDECOMP/PARAFAC decomposition (CPD) [111]. In this dissertation, with the RFoT method, we utilize CPD to extract latent patterns from malware data. CPD compresses the D-dimensional tensor $\mathcal{X}$ into lower-dimensional R rank-1 tensors, also referred to as components. The sum of these R rank-1 tensors approximate the original tensor as follows:

$$\mathfrak{X} \approx \sum_{r=1}^{R} \lambda_r \cdot \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(D)} \tag{2.3}$$

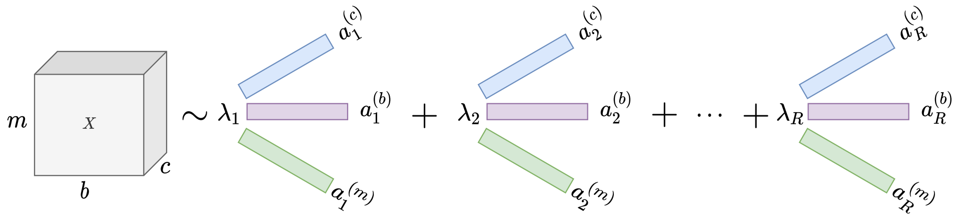


Figure 2.2: Illustration of CPD on a 3-dimensional tensor

Here, $\circ$ denotes the outer product. The latent factors $\mathbf{a}_r^{(d)}$ correspond to each dimension d, where $1 \leq d \leq D$, and the rth component, where $1 \leq r \leq R$, describes the latent information for the given dimension. Each $\mathbf{a}_r^{(d)}$ is normalized to sum up to 1, and the weight is absorbed by each λ_r. An illustration of CPD is provided in Figure 2.2 for a 3-dimensional tensor. The tensor rank R is a hyper-parameter selected by the user. Determining the rank R of a tensor is known to be NP-Hard [111]. CPD can be expressed in a more concise format using the KRUSKAL notation as follows:

$$\mathfrak{X} \approx \mathfrak{M} \equiv [\![\lambda \,;\, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(D)}]\!] \tag{2.4}$$

Here the KRUSKAL tensor $\mathfrak{M}$ is the low-rank approximation of $\mathfrak{X}$. Each

$\mathbf{A}^{(d)}$ is a matrix of latent factors for dimension d. $\mathbf{A}^{(d)}_{:r}$ is the rth latent factor for dimension d with size N_d such that we can write $\mathbf{A}^{(d)}$ as follows:

$$\mathbf{A}^{(d)} = [\mathbf{a}^{(d)}_1, \mathbf{a}^{(d)}_2, \ldots, \mathbf{a}^{(d)}_R] \tag{2.5}$$

Within each latent factor matrix $\boldsymbol{\mathcal{M}}$:: $d - 1 = \mathbf{A}^{(d)}$ for dimension d, linearly dependent columns $\mathbf{A}^{(d)}_{:r}$ can be present for each r [111]. However, when considering all latent factor matrices $\mathbf{A}^{(1,2,\cdots,D)}$ together, the CPD solution is almost always unique [111, 143]. The uniqueness of CPD allows each component to represent distinct events or characteristics of the data. Therefore, the results obtained from CPD provide interpretable outcomes when examining each component individually.

In this dissertation, we employ two popular tensor decomposition algorithms with different properties to heuristically test RFoT's malware classification capability. The first one is the CANDECOMP/PARAFAC Alternating Least Squares (CP-ALS) tensor decomposition algorithm [20, 28, 111]. To fit the tensor $\boldsymbol{\mathcal{X}}$, CP-ALS performs updates using least squares by alternating between each latent factor matrix $\mathbf{A}^{(d)}$, while fixing the remaining $\mathbf{A}^{(d-1)}$ factor matrices until convergence to solve the following optimization function:

$$\min_{\mathbf{A}^{(d)}} ||\mathcal{X} - \mathcal{M}||^2 \tag{2.6}$$

The second tensor decomposition algorithm used in our studies is CANDECOMP/PARAFAC Alternating Poisson Regression (CP-APR), a non-negative tensor decomposition method that minimizes Kullback-Leibler (KL) divergence via an updated Multiplicative Update (MU) algorithm [43]. The CP-APR algorithm includes a non-negativity constraint, which allows the latent factors to be additive parts of the original data, resulting in improved interpretability. In CP-APR, the tensor is modeled under a Poisson distribution with the Poisson rate parameter $\gamma > 0$ as follows:

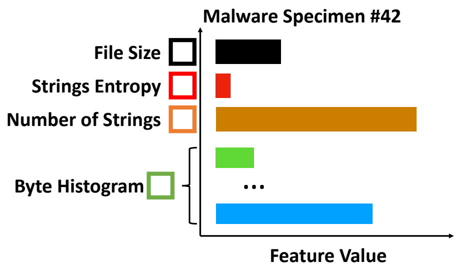
$$\mathcal{X}_{\mathbf{i}} \sim \text{Poisson}(\gamma_{\mathbf{i}}) \tag{2.7}$$

The CP-ALS algorithm was initially included in the widely used MATLAB Tensor Toolbox [22]. With RFoT, we introduce a Python implementation of CP-ALS [1], utilized in our experiments. For the CP-APR algorithm, we utilize an existing Python implementation with GPU capabilities that was previously introduced [61, 72]. For further information on tensors, we

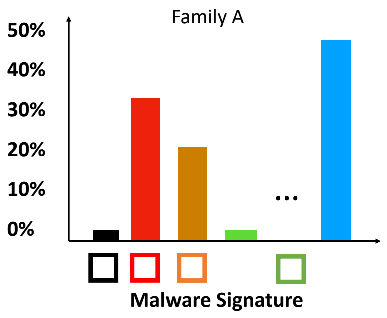
[1]CP-ALS is available at `https://github.com/MaksimEkin/pyCP_ALS`

recommend [111, 147]. More details regarding the CP-APR algorithm can be found in [43], and additional information about CP-ALS is available in [28].

2.0.4 Non-negative Matrix Factorization



(a)



(b)

Figure 2.3: **(a)** This illustration showcases observable data of a malware specimen. The observable features encompass file size, string entropy, number of strings, and byte-histogram of the sample, each with distinct feature values. **(b)** This illustration depicts latent signatures that characterize a particular malware family. These signatures consist of specific feature values summarizing the characteristics of a group of specimens assumed to belong to the same malware family.

The dimensionality reduction method utilized in HNMFk Classifier and MalwareDNA is relies on Non-negative Matrix Factorization (NMF). NMF is an unsupervised learning approach based on low-rank matrix approximation. In our studies, NMF is applied to unfolding of the malware tensor data over the first mode. NMF aims to extract latent (or hidden) patterns from observable data. In comparison to directly observable raw data, in large-scale malware analysis, these latent patterns patterns encapsulate characteristics of a group of malware, enabling downstream decision making tasks such as classification.

For instance, in Figure 2.3-a, we present an illustration of observable features of a malware specimen. When dealing with a dataset consisting of samples exhibiting similar observable features as shown in Figure 2.3-a, NMF's objective is to derive a hidden signature that describes or summarizes the patterns present in this dataset. Figure 2.3-b provides an example of such a signature representing the patterns of a malware family, referred to as "Family A". While observable features of samples may exhibit variations, the signature offers a concise summary of these variations and can effectively identify new incoming samples regardless the small variations in the observable/raw features that do not change the sample's overall characteristic.

NMF represents an observed non-negative matrix, $\mathbf{X} \in \mathbb{R}_{+}^{n \times m}$, as a product of two (unknown) non-negative matrices, $\mathbf{W} \in \mathbb{R}_{+}^{n \times k}$, and $\mathbf{H} \in \mathbb{R}_{+}^{k \times m}$, where usually $k \ll m, n$. Here, m is the number of samples, and n is the number of features. An example illustration of this is provided in Figure

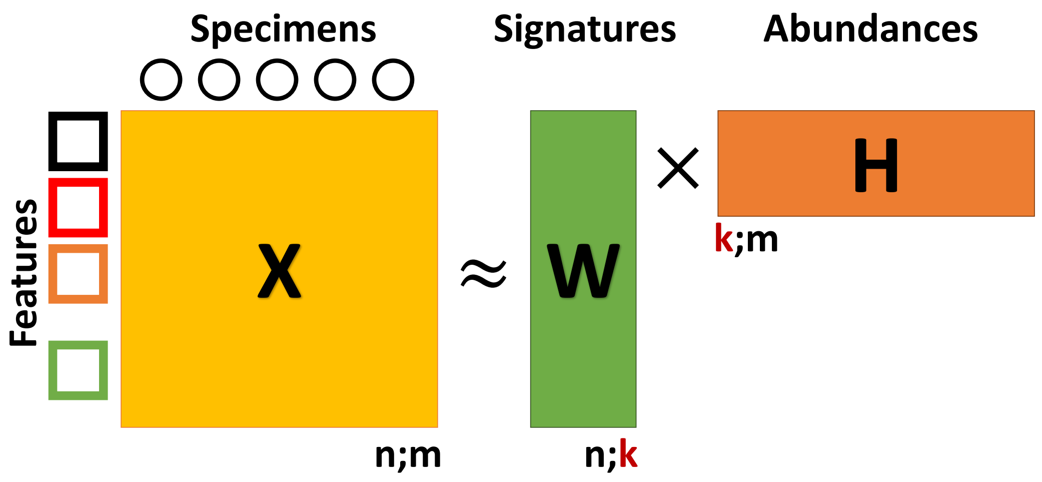


Figure 2.4: This illustration demonstrates how Non-negative Matrix Factorization (NMF) extracts latent patterns from observable data **X**, which includes sample specimens sharing a common feature space. The patterns are encoded within the latent factor **W**, which holds the signatures of features. Meanwhile, the latent factor **H** holds the abundances of these features and clusters the specimens based on these abundances.

2.4. This approximation is performed via non-convex minimization with a given distance, $||...||_{dist}$, constrained by the non-negativity of **W** and **H**: $\min||\mathbf{X}_{ij} - \sum_{s=1}^{k} \mathbf{W}_{is}\mathbf{H}_{sj}||_{dist}$. NMF relies on a generative statistical model predetermined by the choice of the distance $||...||_{dist}$. For example, if the Frobenius norm is chosen as a distance, NMF can be treated as a Gaussian mixture model [82]. If KL-divergence is chosen, we have a generative Poisson model [38], equivalent to latent Dirichlet allocation under uniform Dirichlet prior [51]. In both cases, the number of latent features of the superimposed components is equal to the size of the small dimension k, and NMF minimization is equivalent to the expectation-minimization (EM) algorithm. In this probabilistic interpretation of NMF, the observables are the rows of $\mathbf{X}$ generated by latent variables, the rows of the matrix $\mathbf{W}$, with weights (the basis patterns), represented by the columns of matrix $\mathbf{H}$. Thus, each row $\mathbf{X}_{i:}$ of $\mathbf{X}$ is generated from a probability distribution with mean $\mathbf{X}_{i:} = \sum_{s=1}^{k} \mathbf{H}_{is}\mathbf{W}_{s:}$.

2.0.5 Automatic Model Determination

The NMF minimization requires prior knowledge of the latent dimensionality, k (the number of latent features), which is usually unavailable. It is known that choosing too small a value of k leads to a poor approximation of the observables in $\mathbf{X}$ (*under-fitting*), while if k is chosen to be too large, the extracted features are not easily explainable because they also fit the noise in the data (*over-fitting*). In other words, choosing k is equivalent to estimating the number of parameters of the model, which is a well-known but difficult problem.

In general, the existing partial solutions of this problem are heuristic. Among these solutions is Automatic Relevance Determination (ARD) [123] which was first modified for Principal Component Analysis [30], and then for NMF [128, 162]. Another approach is based on an assumed stability of the

NMF solution, and was proposed to identify the number of stable clusters in the observational matrix $\mathbf{X}$ [36]. A recent model selection technique, called NMFk [11], has been successfully used to decompose the largest collection of human cancer genomes [12]. NMFk integrates classical NMF-minimization with custom clustering and Silhouette statistics [154], and combines the accuracy of the minimization and robustness/stability of the NMF solutions, when a bootstrap procedure (i.e., generation of a random ensemble of slightly perturbed input matrices) is applied to estimate the number of latent features, see for example, [13]. Recently, NMFk was applied to a large number of synthetic datasets with a predetermined number of latent features, and it was demonstrated its superior performance of correctly estimating k in comparison to the other known heuristics [132]. The superior performance of NMFk method as a model selection was also demonstrated in identifying mutational genome signatures in a large set of cancer genomes, both in practice [14] and in large set of synthetic cancer genomes with predetermined number of latent features [98]. In addition, it was shown that NMFk performs better than spherical k-means and other methods for topic extraction [170]. Our numerical experiments in Section 5.0.2 demonstrate that NMFk performs better than the predetermined k=2 case. Therefore, we use NMFk as the core factorization method with automatic model selection needed to extract the "clean" clusters of malware, after the NMF dimension reduction. In this work, we are making extensive use of NMFk, and for completeness we provide the pseudocode for it in Algorithm 1 and a description of it, as follows:

Algorithm 1: NMFk($\mathbf{X}$, k^{min}, k^{max}, M, $Sill\_thr = 0.8$)

```
Require: : $\mathbf{X} \in \mathbb{R}_+^{n\times m}$ , $k^{min}$, $k^{max}$ , $r$
 1: for $k$ in $k^{min}$ to $k^{max}$ do                    ▷ Start and end process for NMFk
 2:     for $q$ in 1 to $M$ do                        ▷ Num. of Perturbations on each k
 3:         $\boldsymbol{\mathcal{X}}_{::q}$ = Perturb($\mathbf{X}$) ▷ Resampling $\mathbf{X}$ to create a random ensemble
 4:         $\boldsymbol{\mathcal{W}}_{::kq}$,$\boldsymbol{\mathcal{H}}_{::kq}$ = NMF($\boldsymbol{\mathcal{X}}_{::q}$,k)
 5:     end for
 6:     $\boldsymbol{\mathcal{W}}^{all}$=[$\boldsymbol{\mathcal{W}}_{::k1}$,. . . ,$\boldsymbol{\mathcal{W}}_{::kM}$] and $\boldsymbol{\mathcal{H}}^{all}$=[$\boldsymbol{\mathcal{H}}_{::k1}$,. . . ,$\boldsymbol{\mathcal{H}}_{::kM}$]
 7:     $\hat{\boldsymbol{\mathcal{W}}}$, $\hat{\boldsymbol{\mathcal{H}}}$ = customCluster( $\boldsymbol{\mathcal{W}}^{all}$,$\boldsymbol{\mathcal{H}}^{all}$)
 8:     $\widetilde{\boldsymbol{\mathcal{W}}}_{::k}$ = medians( $\hat{\boldsymbol{\mathcal{W}}}$)
 9:     $\boldsymbol{\mathcal{H}}^{reg}_{::k}$ = NNLS($\mathbf{X}$,$\widetilde{\mathbf{W}}_{::k}$)  ▷ Column-wise regression of $\mathbf{H}$ with $\widetilde{\mathbf{W}}$ and
    column of $\mathbf{X}$
10:     $\mathbf{s}_k$ = clusterStability($\hat{\boldsymbol{\mathcal{W}}}$)
11:     $\mathbf{err_k}$ = reconstructErr($\mathbf{X}$,$\widetilde{\mathbf{W}}_{::k}$ , $\mathbf{H}^{reg}_{::k}$ )                    ▷ Column-wise
    reconstruction error for L-statistics
12: end for
13: $\mathbf{err}^{all}$=[$\mathbf{err}_{k^{min}}$,. . . ,$\mathbf{err}_{k^{max}}$]
14: $k^{opt}$ = PvalueAnalysis($\mathbf{err}^{all}$ ,$k^{min}$,$k^{max}$,$\mathbf{s}_k$,$Sill\_thr$) ▷ Predicted k value
    using Wilcoxon
15: return $\widetilde{\boldsymbol{\mathcal{W}}}_{::k^{opt}}$, $\boldsymbol{\mathcal{H}}^{reg}_{::k^{opt}}$, $k^{opt}$
  Ensure: $k = k^{opt}$,$\widetilde{\boldsymbol{\mathcal{W}}}_{::k^{opt}} \in \mathbb{R}_+^{n\times k}$ ,$\boldsymbol{\mathcal{H}}^{reg}_{::k^{opt}} \in \mathbb{R}_+^{k\times m}$ , $\mathbf{X} = \widetilde{\boldsymbol{\mathcal{W}}}_{::k^{opt}} \boldsymbol{\mathcal{H}}^{reg}_{::k^{opt}}$
```

1. *Resampling*: Based on the observable matrix, $\mathbf{X}$, NMFk creates an

ensemble of M random matrices, $[\mathcal{X}_{::q}]_{q=1,...,M}$, with means equal to the original matrix $\mathbf{X}$. Each one of these random matrices $\mathcal{X}_{::q}$ is generated by perturbing the elements of $\mathbf{X}$ by a small uniform noise, such that: $\mathcal{X}_{ijq} = \mathbf{X}_{ij} + \delta$, for each $q = 1, ..., M$, where δ is the small error.

2. *NMF minimization*: We use the Frobenius norm-based multiplicative updates (MU) algorithm [116] to explore different numbers of latent features, k, in an interval $[k^{min}, k^{max}]$, for each one of the generated M random matrices.

3. *Custom clustering:* For each $k \in [k^{min}, k^{max}]$, NMF minimizations of the M random matrices, $[\mathcal{X}_{::q}]_{q=1,...,M}$, results in M pairs $[\mathcal{W}_{::kq}; \mathcal{H}_{::kq}]_{q=1,...,M}$. Further, NMFk clusters the set of the $M * k$ latent features, the columns of $\mathcal{W}_{::kq}$. The NMFk custom clustering is similar to k-means, but it holds in each one of the clusters exactly one column from each of the M NMF solutions. This constraint is needed since each NMF minimization gives exactly one solution $\mathcal{W}_{::kq}$ with the same number of columns, k. In the clustering, the similarity between the columns is measured by the cosine similarity metric.

4. *Robust* $\mathbf{W}$ *and* $\mathbf{H}$ *for each k:* The medians of the clusters, $\widetilde{\mathcal{W}}_{::k}$, are

the robust solution for each explored k. The corresponding mixing coefficients $\boldsymbol{\mathcal{H}}^{reg}_{::k}$ are calculated by regression of $\mathbf{X}$ on $\widetilde{\boldsymbol{\mathcal{W}}}_{::k}$.

5. *Cluster stability via Silhouette statistics:* NMFk explores the stability of the obtained clusters, for each k, by calculating their Silhouettes [154]. Silhouette statistics quantify the cohesion and separability of the clusters. The Silhouette values range between $[-1, 1]$, where -1 means an unstable cluster, while $+1$ means perfect stability.

6. *Reconstruction error:* Another metric NMFk uses is the relative reconstruction error, $R = ||\mathbf{X} - \boldsymbol{\mathcal{X}}^{rec}_{::k}||/||\mathbf{X}||$, where $\boldsymbol{\mathcal{X}}^{rec}_{::k} = \widetilde{\boldsymbol{\mathcal{W}}}_{::k} * \boldsymbol{\mathcal{H}}^{reg}_{::k}$, which measures the accuracy of the reproduction of initial data by a given solution and the number of latent features k.

7. *L-statistics:* NMFk uses L-statistics [171] to automatically estimate the number of latent features. To calculate L-statistics for each k, NMFk records the distributions of the column reconstruction errors, $\mathbf{e}_i = \|\mathbf{X}_{:j} - \boldsymbol{\mathcal{X}}^{rec}_{:jk}\|/\|\mathbf{X}_{:j}\|$; $j = 1, ..., m$. L-statistics compares the distributions of column errors for different k by a two-sided Wilcoxon rank-sum test [91], which evaluates whether two samples are taken from the same population.

8. *NMFk final solution:* The number of latent features, k^{opt}, is determined as the maximum number of stable clusters corresponding to a good accuracy of the reconstruction. The Wilcoxon rank-sum test determines the p-value of the given k^{opt}. NMFk is "looking" for a distribution of the column errors such that the next distributions (each one with bigger k) are statistically the same, and the model is fitting the noise. The L-statistics used in conjunction with the condition that the *minimum* Silhouette be greater than 0.80. The threshold of 0.80 is selected to place the predicted k^{opt} prior to a steep decline in the *minimum* Silhouette. The corresponding $\widetilde{\boldsymbol{\mathcal{W}}}_{::k^{opt}}$ and $\boldsymbol{\mathcal{H}}^{reg}_{::k^{opt}}$ are the robust solutions for the low-rank factor matrices.

We provide a sample Silhouette score and relative error plot produced by NMFk for two factorizations, to demonstrate the selection of k, in Figure 2.5. The presented NMFk framework estimates the latent feature count based on two criteria, namely a high minimum Silhouette score, and a low relative reconstruction error, which corresponds to a stable NMF solution. The number of features with lower minimum Silhouette scores correspond to overlapping clusters or scattered clusters. On the other hand, the relative reconstruction error decreases monotonically with the number of latent

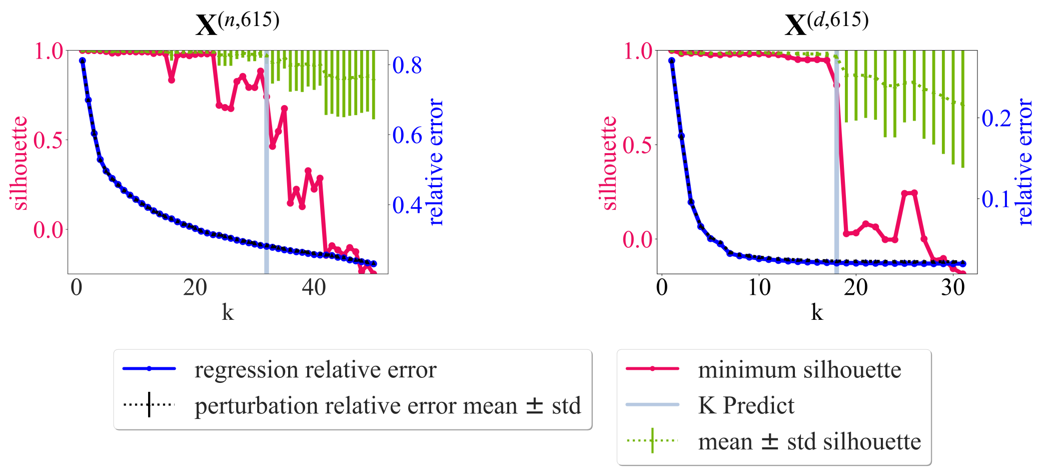


Figure 2.5: Sample Silhouette and relative error graphs obtained from NMFk is shown for the matrices $\mathbf{X}^{(n,615)}$ and $\mathbf{X}^{(d,615)}$ which are formed using 1,000 malware specimens from 10 families. $\mathbf{X}^{(d,615)}$ consist of samples extracted from a single cluster after the the first NMFk procedure on $\mathbf{X}^{(n,615)}$.

features. This decrease is more prominent up to the estimated number of topics followed by a reduced change in the error. As observed in Figure 2.5, with the further increase in the number of latent features past the estimated k, there is a sudden decline in the Silhouette score due to the over-fitting phenomenon as the model tends to fit noise.

In this work, we use the publicly available version of NMFk developed by our team, named Tensor Extraction of Latent Features (T-ELF) [69][2].

[2]T-ELF is available at `https://github.com/lanl/T-ELF`

2.0.6 Hierarchical Modeling

The NMF and Hierarchical NMF [27, 87, 114] strategies have been used successfully for document clustering [39, 177], and topic modeling [67, 73, 86, 156, 169]. For HNMFk Classifier and MalwareDNA we use NMFk to compute clusters of malware specimens by applying it in a hierarchical manner, where successive node expansions focus on the subset of **X** obtained from the parent cluster. Here the clusters are determined using the rows of **H** via *H-clustering* that cluster the specimens' coordinates in the reduced space, also called fuzzy-clustering (i.e. the columns of the matrix **H** with argmax operation) [171]. When going deeper in the graph towards the leafs, we investigate different characteristics of the specimens in the same group, while utilizing the labels as a reference, and achieve better separability of the malware specimens.

Let us consider a simple example from hierarchical document clustering. We assume three well-curated clusters in a text corpus of news articles about sports, technology, and the economy. If we cluster these documents with NMFk and *H-clustering* we can obtain three *"super"* clusters for sports, technology, and the economy. We can further divide the cluster containing sport articles into sub-topics, such as soccer, football, tennis, skiing etc.

by applying additional iterations of NMFk. In our analysis, we choose to select back the specimens corresponding to each one of the super clusters and apply NMFk again. This is the idea behind the hierarchical approach, and consequently a hierarchical approach is used in our methods HNMFk Classifier and MalwareDNA to further separate more heterogeneous clusters based on the known (or labeled) malware instances.

In our semi-supervised setting, the experimental setup contains data with labeled (known/training) and unlabeled (unknown/testing) malware specimens. This allows us to choose a scoring function, not based on information gain (such as normalized discounted cumulative gain from information retrieval [99]) [114] or a fixed threshold using the number of specimens in the cluster [87] to determine which node to take further. Instead, we use a *cluster uniformity score* that measures the stability of the cluster, based on the known specimens in the cluster, as the node expansion criteria for a cluster. In general, the application of NMFk to semi-supervised data will place into each of the final clusters both labeled and unlabeled malware specimens. This allows us to continue to build the hierarchical graph until the further expansion of a particular node is stopped if no unlabeled or labeled samples are present in this node, or if the cluster uniformity score calculated based on the known samples passes the provided threshold.

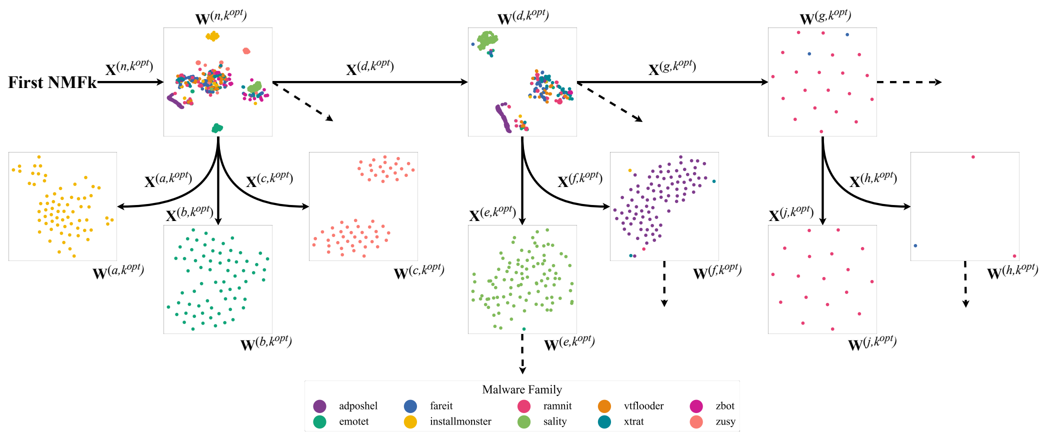

First NMFk
$\mathbf{X}^{(n,k^{opt})}$
$\mathbf{W}^{(n,k^{opt})}$
$\mathbf{X}^{(d,k^{opt})}$
$\mathbf{W}^{(d,k^{opt})}$
$\mathbf{X}^{(g,k^{opt})}$
$\mathbf{W}^{(g,k^{opt})}$
$\mathbf{X}^{(a,k^{opt})}$
$\mathbf{X}^{(b,k^{opt})}$
$\mathbf{X}^{(c,k^{opt})}$
$\mathbf{W}^{(a,k^{opt})}$
$\mathbf{W}^{(b,k^{opt})}$
$\mathbf{W}^{(c,k^{opt})}$
$\mathbf{X}^{(e,k^{opt})}$
$\mathbf{X}^{(f,k^{opt})}$
$\mathbf{W}^{(e,k^{opt})}$
$\mathbf{W}^{(f,k^{opt})}$
$\mathbf{X}^{(j,k^{opt})}$
$\mathbf{X}^{(h,k^{opt})}$
$\mathbf{W}^{(j,k^{opt})}$
$\mathbf{W}^{(h,k^{opt})}$
Malware Family
adposhel
emotet
fareit
installmonster
ramnit
sality
vtflooder
xtrat
zbot
zusy

Figure 2.6: The path of the hierarchical graph formed by the NMFk is shown using 1,000 malware specimens containing a total of 10 malware families. After each factorization, the clustering is visualized by reducing the dimensions of $\mathbf{W}$ using t-SNE. Dashed arrows are used to indicate the existence of an another sub-tree from the node. Since we are obtaining k_{opt_i} subsets of specimens from current $\mathbf{X}$ at each stage, $n > a \geq b \geq c \geq d > e \geq f \geq g > j \geq h$.

We provide an example visualization of the latent factors obtained from NMFk with a hierarchical setting in Figure 2.6. Note that on this specific example *W-Clustering* (apply clustering over the columns of $\mathbf{W}$ is used, instead of *H-Clustering*). Here, we apply dimensionality reduction using *t-SNE* [168] to each latent factor $\mathbf{W}$ to plot the clusters. Each point in the embedding of $\mathbf{W}$ is colored based on the family to which the specimen belongs. Here the clusters are expanded until all the samples in the cluster belong to a single class. The *t-SNE* visualization show how the hierarchical clusters of malware families are formed, and how the clusters become more homogeneous as we perform additional applications of NMFk.

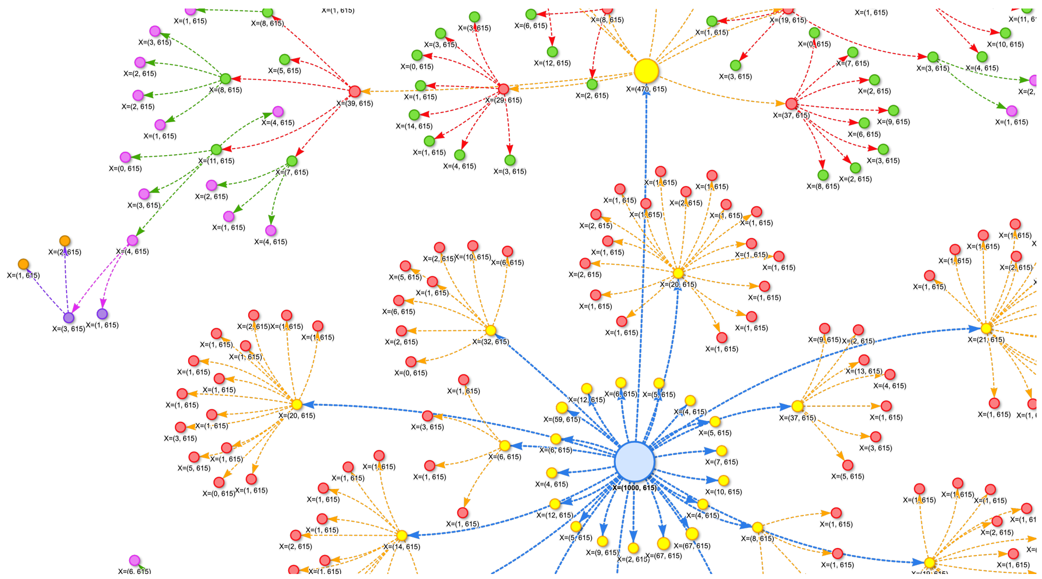


Figure 2.7: This visualization showcases a segment of a graph generated by hierarchically applying NMFk to malware data. It emphasizes the comprehensive use of NMFk to separate mixed signatures, resulting in a graph with a topological structure where samples are progressively partitioned until a distinct signature is identified.

Figure 2.6 demonstrates a relatively small application of the hierarchical approach, while Figure 2.7 further showcases the graph produced by this method. Although Figure 2.7 displays only a small segment of the entire graph, it is apparent that the graph is extensive and structured in a topology where NMFk is applied multiple times within each subgraph until a unique signature is obtained. Additionally, each subgraph, or depth level, operates independently of the others. This independence allows the NMFk operations performed at each node to be executed on a High Performance Computing (HPC) system in an embarrassingly parallel manner. This parallelization is helpful in accelerating the computation within the hierarchical approach. We introduce our HPC considerations in Section 6.0.4.

2.0.7 Datasets and Pre-processing

In our experiments, we use the EMBER-2018 dataset [15] for the frameworks RFoT and HNMFk Classifier. In addition to the EMBER-2018 dataset, we incorporate an additional dataset named MOTIF [103] for experiments involving the MalwareDNA framework. This subsection will summarize these two datasets and detail the preprocessing steps applied to our data.

EMBER-2018 Dataset

Collection of malware data has challenges such as copyright issues, labeling difficulty, and security precautions. Therefore, compared to other ML fields with abundant data (such as text and images), the malware identification community has lacked a benchmark dataset sufficient to enable reproducibility and comparison of new methods. To address this issue, Anderson et. al. released the EMBER-2018 dataset [15]. Since its release, EMBER-2018 has become a popular benchmark dataset for ML-based malware analysis methods.

Set	**Families**	**Samples**	**Novel Families**
Known (Default Train)	2,730	289,026	1,982
Unknown (Default Test)	916	99,216	168

Set	**Min Family**	**Max Family**	**Samples/Family**
Known (Default Train)	1	16,689	105.8
Unknown (Default Test)	1	19,260	315.5

Table 2.2: EMBER-2018 dataset default train and test set split and malware family and sample counts are displayed. Novel families for the known (or train) set are the families that only exist in the training set. The novel families for unknown (or test) set are the families that only exist in the test set (i.e. we do not see these families during inference, or we do not have known specimens for reference). Min Family and Max Family columns show the minimum and maximum number of samples exist for a family in the dataset. For instance, there are malware families with single sample in both known and unknown sets. Samples/Family column shows the average number of samples per family.

EMBER-2018 is a collection of PE header and meta-data information extracted from 1.1 million benign and malicious Microsoft Windows Portable executable binaries, out of which 800,000 have labels.

One advantage of using the EMBER-2018 dataset is that the distribution of the family classes resembles real-world cases. The training portion of the dataset contains malware families that do not exist in the test portion of the data. Similarly, the test set contains novel malware families, or the malware families that do not exist in the training set. This is also shown in Table

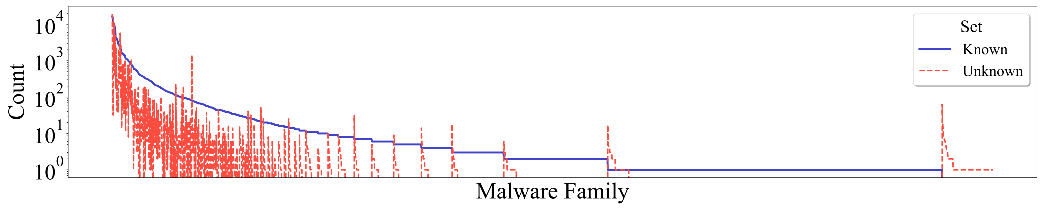


Figure 2.8: Distribution of the malware families in EMBER-2018 dataset. Count of family classes are shown in log scale for both the training and test set. Both the training and test sets has an extremely imbalanced classes of malware families, and the test set of contains set of novel malware families.

2.2. 1,982 of the malware families, making over 11 thousand samples, are not seen again in the test set. There are 168 novel families, forming 363 samples, that we do not have any reference of in the training set. At the same time, malware family classes in EMBER-2018 are extremely imbalanced. Figure 2.8 shows the distribution of the malware families for both the training and test set. For instance, there are malware families that consist of single samples, including the specimens from the novel families (which can also be seen at the right side of Figure 2.8 with red-dashed line). In fact, the majority of the malware families in the dataset consist of less than 10 samples. We next proceed to the pre-processing of the features to remove the outliers.

During our experiments, we represent each file in the dataset as a collection of features from both general file meta-data as well as PE header information. Each of the features are concatenated vertically to form the final features matrix. This is equivalent to forming an 11 dimensional tensor, with the dimensions *Samples* × *Feature 1* × *Feature 2* × ... × *Feature 10*, and taking the mode-1 unfolding of the tensor. Specifically, we use the following features:

1. *byte histogram*: a vector of size 256 where each entry represents the number of times a certain byte occurs in the file.

2. *byte entropy*: normalized joint distribution of entropy and byte values.

3. *print table distribution*: distribution of characters obtained from printable strings with minimum of 5 consecutive printable characters in the binary.

4. *strings entropy*: measure of randomness of printable strings present in the malware.

5. *number of strings*: number of printable strings.

6. *file size*: size of the binary in bytes.

7. *number of exports*: number of functions exported by the malware.

8. *number of imports*: number of functions imported by the malware.

9. *code size*: size of *.text* or code section of the PE header in bytes.

10. *number of sections*: number of sections present in PE header.

Our dataset consists of heterogeneous features containing outlier values. Since NMF is susceptible to outliers (extremely large or small values in the rows of initial data), see for example [182], we normalize the features used in our analysis. This normalization prevents the larger values in the rows of the initial data to bias/skew the NMF optimization procedure by favoring some of the columns in $\mathbf{X}$, see details in Ref.[98]. Note that the case of outliers affecting NMF optimization is distinct from the characteristics makeup of a novel malware family: After the normalization, the novelty of the malware belonging to unknown (or never seen before) family reflects on the shape of its latent signature (the columns of matrix $\mathbf{W}$).

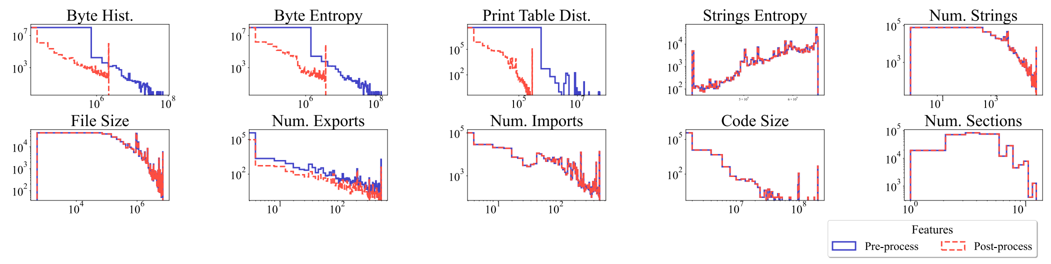


Figure 2.9: Static malware analysis based features from PE header files and malware meta-data used in the analysis shown before and after the mapping of the outliers, defined by $Z = 3$ statistical score, for both training (known specimens) and test sets (unknown specimens).

In our normalization, Z-scores are used to remap the outliers that are more than or less than 3 standard deviations away from the mean. These outliers are mapped to the point that is exactly 3 standard deviations away from the mean. In Figure 2.9, we show the histogram of feature values for pre- and post-processing. The normalization was most prominent among the features *byte histogram*, *byte entropy*, and *print table distribution*. Finally, we scale the values to be between 0 and 1.

MOTIF Dataset

For the experiments involving the MalwareDNA framework, we also use the Malware Open-source Threat Intelligence Family (MOTIF) dataset [103]. MOTIF comprises 3,095 malware samples distributed across 454 families. In contrast to MOTIF, the family labels in the EMBER-2018 dataset are determined using AVClass, resulting in *weakly labeled* classes due to the inherent inaccuracies in AVClass's family labeling [185]. AVClass occasionally retains generic family names, shows inconsistency in the use of aliases for malware families, and is susceptible to errors in antivirus signatures which can affect its accuracy. Despite these imperfections, AVClass remains one of the accepted methods for obtaining a large quantity of malware family labels.

In comparison, the MOTIF dataset provides ground truth family labels, offering a more reliable source for this aspect of the analysis. We process the MOTIF dataset using the same static-analysis-based features and apply identical normalization techniques as those used with the EMBER-2018 dataset.

2.0.8 Experimental Setups and Baselines

In this section we provide a summary of our experimental setup as well as describe our baseline models.

Experimental Data Setups

In this dissertation, we employ various experimental setups, which are provided in this section and are also summarized in Table 2.3. Specifically, in Section 4.0.5, where we present our analysis for the RFoT framework, we randomly sample both benign and malicious software specimens from the EMBER-2018 dataset. This sampling is conducted ten times, each consisting of 10,000 specimens. For RFoT experiments, we used 30% of 10,000 malware and benign instances for each 10 subsets as testing or unknown set.

Similarly, the EMBER-2018 dataset is utilized for the analysis involving

Method	Section	Dataset	Size	Benign-ware	# Families	# Novel Family	Class-imbalance
RFoT	4.0.5	[15]	10K	✓	Random	—	—
HNMFk Classifier	5.0.2	[15]	10K	—	10	—	—
HNMFk Classifier	5.0.2	[15]	10K	—	10	—	—
HNMFk Classifier	5.0.2	[15]	10K	—	10	—	—
HNMFk Classifier	5.0.2	[15]	388k+	—	2,899	168	✓
HNMFk Classifier	5.0.2	[15]	10K	—	10	—	—
MalwareDNA	6.0.5	[15]	1K-20K	✓	4	1	—
MalwareDNA	6.0.5	[15]	10K	—	7	1	✓
MalwareDNA	6.0.5	[15]	10K	—	7	1	✓
MalwareDNA	6.0.5	[103]	529	—	8	1	—
MalwareDNA	6.0.5	[15]	15K	✓	7	1	—

Table 2.3: Experiment setups are summarized for each method, alongside the corresponding section number. The **Dataset** column lists the dataset used in each section for the specified method. Citation [15] for the **Dataset** refers to EMBER-2018 while the citation [103] refers to the MOTIF dataset. The **Size** column provides the total size of the dataset. A check-mark in the **Benign-ware** column indicates that benign samples were included in the analysis. The **# Families** column lists the number of malware families included in the dataset, where "Random" denotes that the malware was sampled randomly and includes a mix of different families. Finally, the **# Novel Family** column specifies the number of malware family classes that are present only in the unknown or testing set for analysis. "—" indicates that the given column or experimental setup consideration is not taken for the method in the provided section.

the HNMFk Classifier. For the experiments assessing the performance of the HNMFk Classifier, we use a smaller subset of the data under unrealistic conditions. This subset comprises 10,000 specimens selected from the top 10 most populous malware families, with 1,000 specimens per family. These experiments are detailed in Sections 5.0.2, 5.0.2, 5.0.2, and during the ablation studies for the HNMFk Classifier in Section 5.0.2. For these tests, the data is randomly sampled 10 times.

Furthermore, we use the entire EMBER-2018 dataset—excluding the benign specimens—under realistic conditions to evaluate the HNMFk Classifier, as discussed in Section 5.0.2. In this more extensive experiment, the dataset is predefined with default training and testing splits of EMBER-2018: the training set contains over 289,000 specimens from 2,730 malware families, while the test portion includes approximately 99,000 samples from 916 malware families. Details of the dataset sample sizes and family statistics are provided in Table 2.2.

For MalwareDNA, we employ several different experimental setups. Initially, using the EMBER-2018 dataset, Section 6.0.5 presents the preliminary results for MalwareDNA, where we randomly sample 1,000 benign and malware specimens from four specific families: Ramnit, Adposhel, Emotet, and Zusy. In these experiments, Ramnit is chosen to represent the novel malware family.

We then expand these experiments by increasing the sample size to 20,000 specimens. This sampling is conducted once and does not include confidence intervals for the results. To address this, subsequent experiments involve randomly sampling 10,000 malware specimens ten times from the EMBER-2018 dataset, focusing on the top seven populous malware families: Ramnit, Adposhel, Emotet, Fareit, InstallMonster, Xtrat, and Zusy. Ramnit is again

Malware Family	**Training Set**	**Testing Set**
xtrat	4853.9 (+- 12.6)	543.1 (+- 12.2)
installmonster	3750.3 (+- 10.2)	416.7 (+- 11.5)
adposhel	3216.4 (+- 6.6)	361.6 (+- 5.6)
zusy **(rare family)**	638.0 (+- 7.0)	67.0 (+- 6.9)
emotet **(rare family)**	232.2 (+- 3.8)	25.8 (+- 3.8)
fareit **(rare family)**	97.2 (+- 1.9)	11.8 (+- 1.4)
ramnit **(novel family)**	0.0	1029.0 (+- 2.4)

Table 2.4: Distribution of malware families in training and testing sets reported with mean number of instances and the confidence interval over 10 sample trials. This setup is used for MalwareDNA experiments in Sections 6.0.5 and 6.0.5.

selected to represent the novel family, particularly when assessing the confidence metrics for the reject-option in Section 6.0.5, and when demonstrating the performance of MalwareDNA under class imbalance in Section 6.0.5. For the class-imbalance experiments, with the increasing under-sampling-rate we under-sample the families randomly ten times Zusy, Emotet, and Fareit to highlight the rare families (i.e. Fareit is the rarest class). We summarize the dataset used in the class-imbalance experiments in Table 2.4. This same dataset was also used to test our confidence methods in the aforementioned Section 6.0.5. Finally, the capabilities of MalwareDNA for distinguishing benign samples are showcased in Section 6.0.5 where we randomly sample the same seven malware families ten times, select Ramnit as the novel family, and also include benign specimens in the sampling, extending the data size to 15,000.

Meanwhile, in the experiments that utilize the MOTIF dataset, as described in Section 6.0.5, we focus on the top eight most populous malware families from MOTIF dataset and select one of the families randomly to represent the novel specimens. We randomly sample these families ten times to create datasets, each consisting of about 500 specimens.

Experimental Baseline Setups

We compare our method against the well-established supervised malware classifiers, XGBoost [42] and LightGBM [108]. Additionally, we enhance these baselines by incorporating the SelfTrain algorithm [179] to develop semi-supervised models. Previous studies have benchmarked these models against the EMBER-2018 dataset [15, 124]; however, our experiments challenge these models further by requiring them to classify malware families under conditions of extreme class imbalance and to detect novel malware families simultaneously. Our baseline models are tuned using Optuna [9] across between 50 to 200 trials, employing between 3 to 5-fold stratified shuffle cross-validation to ensure robustness and reliability in our evaluations. The variations in the Optuna settings across different experiments are driven by the distinct computational demands of each case. Consequently, in each section that presents our experimental results, we revisit the tuning and dataset considerations. This repeated emphasis ensures a thorough understanding of the adjustments made and aligns the experimental setups with their respective computational requirements. In our experiments with HN-MFk Classifier, we also include a Multi-Layer Perceptron (MLP) [90] model as a baseline in Section 5.0.2. We tune this MLP baseline with *HyperBand Tuner* [117].

In experiments where we randomly resample the dataset, we report our results along with a 95% confidence interval (CI) for the ten runs. This statistical approach provides assurance about the reliability and consistency of our findings.

2.0.9 Performance Evaluation Metrics

We utilize several traditional machine learning performance metrics, including F1-score, precision, and recall, to evaluate our results. Additionally, we employ the Area Under the Curve of Risk-Coverage (AURC) in experiments where the reject-option was enabled [52]. This section provides a summary of these metrics.

F1, Precision, Recall

To evaluate the performance of our method and the baseline models, we use Precision, Recall, and F1 score. Precision score measures the ability of the model's correctly identify the positive class and can be calculated as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.8}$$

where TP is the number of true positive predictions, FP is the false positives. The Recall metric measures the extent to which model can detect the positive class, and it is calculated as follows:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.9}$$

where FN is the number of false negatives. F1 score is calculated using both the Precision and Recall scores together; therefore, F1 score is only high when both Precision and Recall are high. F1 score can be calculated as follows:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.10}$$

More specifically, F1 can be calculated as follows:

$$F_1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{2.11}$$

Area Under the Risk Coverege (AURC)

The performance of our method, when the reject-option is enabled, is quantified using the Area Under the Curve of Risk-Coverage (AURC) [52]. The Risk-Coverage curve illustrates the trade-off between coverage (the number of samples for which a non-abstaining prediction was made, i.e., non-reject predictions) and risk, which is measured using 0/1-loss or $1-$*score-metric* (the *score-metric* could be, for instance, F1-score or accuracy).

The AURC score, which ranges from 0 to 1, is a metric where a lower AURC is preferred as it indicates reduced risk at any given level of coverage. This curve provides a clearer picture of the trade-offs between model performance and the proportion of inferences where the model either made a decision or abstained. Analysts or users of our models can use this curve to select the optimal balance between risk (model performance) and coverage according to their specific requirements.

Chapter 3

Related Work

Here we give a summary of relevant work pertaining to large-scale malware analysis, class-imbalance and detection of novel malware families, and use of dimensionality reduction techniques in malware analysis and cyber security.

3.0.1 Large-scale Malware Classification

ML-based automated detection and characterization of malware has been a longstanding area of research. Deep learning has proven to be an effective method for classifying malware in a supervised manner. Vinayakumar et al. utilized shallow neural networks to detect malware using PE header features

[146]. Similarly, Fabian et al. employed neural networks, designing their method for use in environments with limited computational resources [83]. In contrast to these approaches that utilize a selected set of static malware features, Raff et al. introduced a deep learning architecture named MalConv, aiming to classify malware directly based on the entire raw byte-sequences of the binary [149]. These methods leverage static malware analysis-based features for malware classification. While static malware features can effectively identify malicious files, dynamic malware features can offer additional insights into the executable. Vinayakumar et al. adopt a multi-modular approach using Deep Neural Networks (DNN), performing classification using features from static analysis, dynamic analysis, and gray-scale malware images [172]. In our work, we take the prior approach for malware with static analysis based features and use PE malware features in our analysis.

Several prior studies have focused on malware detection using images generated from malware [119, 120, 130, 178, 181]. This includes research on classifying malware visualizations utilizing an ensemble of random forests [153], as well as another study employing a semi-supervised approach to cluster gray-scale malware images [6]. Additionally, Wang et al. utilized a semi-supervised approach with gray-scale malware images [173]. In their research, byte n-grams were translated into fixed-size gray-scale image vectors as training features. Wang et al. perform classification with the gray-scale image vectors using a 1-dimensional Convolutional Neural Network (CNN), a supervised deep learning method, that is strengthened using a semi-supervised Generative Adversarial Network (SGAN).

Previous studies have also examined classical tree-based ML methods for malware classification. Kumar et al. demonstrated that XGBoost is an effective model for classifying Windows PE malware, achieved through low-resource feature selection [115]. Pham et al., also using the EMBER-2018 dataset, illustrated that statistical summaries of the original PE features can enhance detection results. They employed LightGBM, which surpassed the previously introduced deep learning solution MalConv while requiring fewer resources [141]. In our experiments, we also test our model against EMBER-2018, benchmarking it against XGBoost and LightGBM, which are considered state-of-the-art baseline models on the EMBER-2018 dataset. While these methods are supervised solutions, we explore semi-supervised learning for its superior generalization capability.

The majority of the aforementioned work, which reported excellent malware detection capabilities, is based on supervised learning. However, supervised models often encounter performance degradation in production when confronted with specimens that do not conform to the same distribution observed during training. Qi et al. tackled this issue by integrating an unsupervised domain adaptation technique based on adversarial learning into LightGBM for static malware detection [142]. Their approach extends LightGBM to learn domain-invariant features by using the predictions generated from each decision tree in the model as a feature space, subsequently employing these features as input into the adversarial learning framework.

Another area that has garnered increasing interest is the application of ensemble learning to augment the predictive capabilities of malware classifiers. In pursuit of this, previous research has delved into an ensemble approach for Windows malware classification utilizing static features. Atluri demonstrated that various tree-based ensemble models, such as Random Forests, Bagging Decision Tree Classifier, and Gradient Boosting Classifier, among others, can be used together in a single framework, named Voting Ensemble Classifier (VEC), to achieve enhanced detection of Windows PE malware [18]. Similarly, Ramadhan et al. explored a comparable method by creating a voting-based ensemble model employing LightGBM, XGBoost, and Logistic Regression [151]. Their study showed that an ensemble of classifiers, each with its distinct inductive biases, could result in increased accuracy compared to any individual model alone, as each member of the ensemble complements the weaknesses of others. Additionally, the framework of ensemble learning has been applied in the realm of deep learning for malware detection by Dahl et al. [46]. The authors demonstrated that an ensemble of neural networks employing voting, alongside a novel feature selection method based on dimensionality reduction and random projections, significantly improves malware identification.

While the aforementioned prior work on ensemble learning utilized the voting method, Azeez et al. adopted a stacking approach employing an ensemble of CNNs to create a derived dataset based on the decisions made by the base models. This derived dataset was then used as input to the final prediction layer, incorporating an ExtraTrees classifier to enhance prediction accuracy [19]. Similarly, Gupta et al. also applied a stacking approach employing an ensemble of diverse supervised classifiers [88]. However, in contrast, Gupta et al. carefully selected the best-performing classifiers within the ensemble by initially ranking them based on their performance. Subsequently, the highest-ranking base models were integrated into the stacking ensemble layer, further enhancing the capability for malware detection. Inspired by the success of ensemble learning, we incorporate an ensemble based approach to RFoT. RFoT leverages an ensemble framework to randomly select a set of tensor configurations, where the members of the tensor dimensions and entries randomly selected from the malware feature-set, and the tensor rank is selected randomly for each member of the ensemble.

As part of the semi-supervised scheme, all of our methods leverages clustering and similarity scores for the categorization of novel samples and classification of malware and malware families. Clustering has been integrated into ensemble learning frameworks for malware identification as well. Ye et al. presented a hybrid framework that constructs base clusters from an ensemble of clustering algorithms separately applied to Term Frequency-Inverse Document Frequency (TF-IDF), built from instruction frequency and instruction n-grams [180]. This approach utilizes an ensemble of clustering algorithms with distinct characteristics, such as hierarchical clustering and weighted subspace K-medoids, to form the base clusters. These clusters are then utilized to extract the signatures that differentiate malware families. Similarly, Zhang et al. proposed a similar framework based on an ensemble of hybrid clustering algorithms [186].

Similar to clustering ensembles, distance metrics have been incorporated within an ensemble learning structure. Kong et al. utilized similarity metrics between pairs of malware to categorize malware families [112]. The authors derived similarity measurements using a distinct set of features such as opcodes, system calls, and file system activity. These new distance-based feature vectors, each derived from different malware features, were then employed to train an ensemble of classical ML models. Furthermore, a similarity-based approach was previously employed by Raff et al., where they introduced the Burrows Wheeler Markov Distance (BWMD), an efficient similarity metric. This metric is based on embedding data into a fixed-size vector space, demonstrating its effectiveness in clustering malware [150]. Finally, the malware similarity for clustering IoT malware in an unsupervised manner was presented in [25].

Reference	Dataset(s)	Dataset Size	Num. Classes	Imbalanced Data	Novel Malware	Method
Ours	EMBER-2018 [16]	388k	2,898	✓	✓	Semi-supervised
[94]	*Custom*	6.5m	100	✓	—	Supervised
[100]	Drebin [17]	5k	40	—	—	Supervised
[172]	Malimg [131] & *Custom*	9k & 10k	25 & 10	—	—	Supervised
[25]	*Custom*	10k	14	✓	—	Unsupervised
[183]	EMBER-2018 [16]	750k	21	—	—	Supervised
[121]	EMBER-2017 [16]	500k	21	✓	—	Supervised
[160]	VirusShare [78]	2.7k	12	—	—	Supervised
[8]	Malimg [131]	21k	9	—	—	Supervised
[127]	*Custom*	115k	8	✓	✓	Supervised
[89]	*Custom*	31k	5	—	—	Supervised

Table 3.1: The comparison of prior and our work in dataset size, number of classes, consideration of imbalanced data and novel malware families, and the method used. *Custom* refers to the proprietary datasets, or the custom build datasets by the authors.

Several previous works have looked at malware family classification, however, they tend to use only the most common malware families, did not consider novel malware families, or used manually balanced datasets when reporting their results [8, 25, 80, 89, 100, 121, 160, 172, 183]. In contrast, when comparing to the baseline models, we report our results when classifying specimens belonging to the whole ensemble of malware families present in the EMBER-2018 dataset with an imbalanced setup which also includes novel unknown specimens for HNMFk Classifer in Section 5.0.2. We further incorporate these considerations in our analysis for MalwareDNA in Section 6.0.5, where we showcase the model's performance when detecting novel malware families and when working under class-imbalance. This setup allows our results to be more like what malware analysts would see in the real world.

It has already been shown that the class-imbalance problem degrades the performance of popular methods developed for large-scale malware analysis [155]. Rajvardhan et al. used BERT to classify imbalanced malware data with high accuracy [135]. While this work only focused on malware/benignware classification, our aim is to classify malware families. Several other prior works also considered class imbalance, however, they still targeted a small number of top malware families, and rare specimens are mapped to a single *"others"* class [121, 127]. To the best of our knowledge, the most realistic and the largest malware family classification work was done by Huang et al. [94], which targeted 100 classes where two of the classes include the one for benign samples and another for the rare specimens. This type of setup, although it considers class imbalance, limits the classification capabilities to only a handful of malware families. In contrast, we do not map the rare specimens into a single class, but rather recognize all 2,898 malware families as individual classes in our results presented in Section 5.0.2. Furthermore, supervised methods used in prior work often poorly generalize to rare specimens as also pointed out by Loi et al [121]. Loi et al. reports that their false positives are heavily represented by the families collected within the *"others"* class due to the supervised method's inability to learn the patterns of these families from a rare number of specimens. We use a semi-supervised approach, which has an added benefit of improved generalizability and ability to work with a low quantity of labeled data compared to the supervised models. In other words, our methods HNMFk Classifier and MalwareDNA do not require training with rare specimens, since it possesses the abstaining prediction ability i.e. the *reject-option*). This allows our method to combine the abilities of malware family classification under class imbalance and novel malware family identification where we make an increased number of abstaining predictions (lower coverage-rate) to maintain the performance or accurate decisions.

A number of these prior works did consider benign-ware as a class in their analysis [121, 127, 172], but we assume the samples are already known to be malware and perform only malware family classification in experiments with HNMFk Classifier in Section 5.0.2. We did include benign specimens in our experiments showcasing preliminary results of MalwareDNA in Section 6.0.5 and our ultimate analysis for MalwareDNA in Section 6.0.5. We summarize the highlights of the mentioned prior work and show how they compare to our research in Table 3.1.

While several other works have examined supervised approaches [8, 89, 94, 100, 121, 127, 160, 172, 183], we draw inspiration from previous advancements and successes in ensemble learning and clustering methodologies. Our framework leverages ensemble learning for RFoT and clustering techniques in multidimensional analysis through tensor decomposition for HNMFk Classifier and MalwareDNA. This approach combines the potency of tensor decomposition with clustering, and with ensemble learning for RFoT. Furthermore, driven by the anticipated advantages, we structure our tensor decomposition-based solutions within a semi-supervised methodology.

3.0.2 Dimensionality Reduction for Malware Analysis

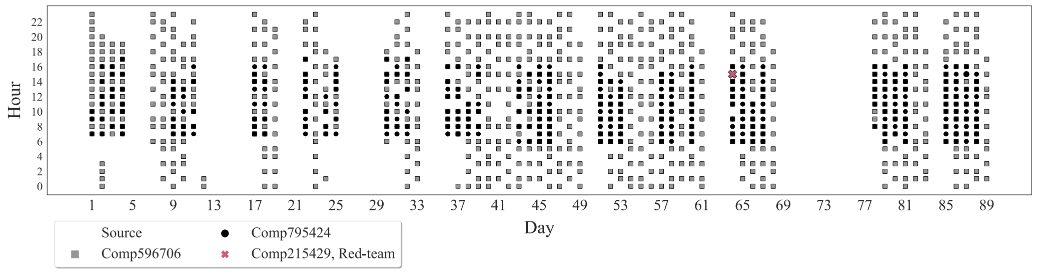


Figure 3.1: Hourly authentication events from multiple source computers over 90 days for one compromised user, User087542, in the LANL Unified Host and Network Dataset [167]. The user's activity reveals time- and device-based predictable patterns that deviate from the single anomalous log-on.

Data relevant to cybersecurity problems often exhibit a multi-dimensional nature, making tensors an ideal tool for analyzing cyber data. Several prior works have employed tensor decomposition to address cybersecurity issues in an unsupervised manner. The CPD has emerged as a popular tool for identifying various types of outliers or anomalies in cyber data [113, 125].

Bruns et al. utilized non-negative tensor decompositions, particularly the CP-APR algorithm, to detect patterns of malicious network activity [37]. In this work, the authors leveraged the interpretability of tensor decomposition results by visually analyzing the latent factors to successfully identify distinct stages of a cyber breach, including reconnaissance, brute-forcing, data exfiltration, and insider threats. Another work integrated CP-APR with a statistical framework which demonstrated effectiveness in enhancing automatic anomaly detection capabilities. This method produced state-of-the-art results in identifying compromised user credentials, botnet network traffic, spam emails, power grid and Supervisory Control and Data Acquisition (SCADA) system anomalies, and fraudulent credit card transactions [58, 72, 129]. Moreover, CP-APR was employed to detect cyber anomalies by utilizing HPC resources for conducting embarrassingly parallel graph analytics within the latent components [79]. In [72], botnet network traffic from the UGR'16 dataset [122][1] was identified using non-negative tensor decomposition within a statistical framework based on user behavior analysis. Users and devices in a network create predictable patterns in time, see for example Figure 3.1, which can be modeled with tensor decomposition. Here, the patterns of normal or expected network traffic were modeled with CP-APR. During testing, this model, utilizing latent components extracted by factorizing the data from expected behavior, identified deviations from the norm via Poisson p-values, where lower p-values were used as an indicator of an anomaly or botnet network traffic. The analysis incorporated IP addresses of the network communication and temporal information

[1]The UGR'16 dataset is available at `https://nesg.ugr.es/nesg-ugr16/`

as tensor dimensions. The study showcased how the unsupervised tensor decomposition-based method surpasses the anomaly detection capabilities of several state-of-the-art supervised and semi-supervised approaches.

The Tucker tensor decomposition has also emerged as a popular algorithm in addressing cybersecurity issues. Kanehara et al. utilized non-negative Tucker tensor decomposition with thresholding across latent factors to enable real-time botnet detection within the darknet [105]. Similarly, Tork et al. employed Tucker tensor decomposition on a three-dimensional tensor to identify telecommunication anomalies [81].

Xie et al. introduced a tensor truncating algorithm for rapid low-rank Tucker decomposition of tensors. They utilized reconstruction error as a metric for detecting network anomalies [176]. Additionally, Sun et al. tackled the network anomaly detection problem by employing a dynamic Tucker tensor decomposition approach tailored for handling large-scale streaming data [161].

For novel malware detection, the work by Cordonsky et al. [45] bears the closest resemblance to ours. In their study, they employ two deep neural network models. Initially, the first model is trained as a multi-class classifier for fourteen malware families. Then, the softmax output layer, or the prediction head, of this trained deep neural network is removed, and the last hidden layer is repurposed for ”dimensionality reduction” to represent the signatures of malware specimens. They hypothesize that this layer will produce lower activation levels for malware families it has not previously seen. Both the known and unknown malware families are then processed through this network, utilizing the last hidden layer to represent their signatures. A second, smaller neural network is trained on these signatures from both known and unknown or novel malware families, aiming for the detection of novel malware in a binary classification framework. However, based on our best understanding of this work, their experimental setup appears to include unknown families in the training phase, specifically for the smaller second model. A more realistic experimental configuration would exclude these unknown families from the training phase. In contrast, our experiments with the HNMFk Classifier and MalwareDNA ensure that the model remains unaware of novel malware family data during training. Additionally, Cordonsky et al. demonstrate the performance of their methods over a balanced dataset. However, in real-world where these models are deployed, they are exposed to unbalanced data where both rare and prominent malware is seen. Our experiments, conversely, engage with unbalanced data, demonstrating how the HNMFk Classifier remains effective under extreme class imbalance, and with MalwareDNA we highlight the limitations of baseline supervised and semi-supervised models in classifying rare malware families due to insufficient training data. MalwareDNA achieves accurate predictions for both rare and prominent families. Lastly, while Cordonsky et al. limit their results to fourteen malware families, our implementation of the HNMFk Classifier extends to classifying over 2,800 malware families.

In classical ML, ensemble learning has proven to significantly enhance individual model capabilities. Motivated by this, there has been growing exploration of ensemble learning in the tensor domain. Kisil et al. were among the first to apply an ensemble learning approach in multi-dimensional space [109]. They utilized an ensemble of latent factors extracted via tensor decomposition to train classical ML models, combining hypotheses from each tensor decomposition and ML model.

Similarly, Hou et al. introduced a framework employing tensor decomposition in an ensemble setting to classify Android malware [93]. This involved utilizing application permissions, API calls, and hardware specifications as features to construct a tensor. Tensor Filter was then employed in a recursive boosting approach to generate an ensemble of base models. Their study demonstrated that the ensemble approach, coupled with tensor decomposition, enhances malware detection capabilities compared to classical ensemble models.

Another prior work also utilized an ensemble approach to demonstrate how anomaly detection capabilities can be enhanced, showcasing that an ensemble of tensors with varying ranks can augment anomaly detection capabilities using statistical p-value fusion techniques [72].

The outcomes derived from the aforementioned prior studies heavily depend on the selection of tensor rank, dimensions, and entries. Crafting an appropriate tensor that yields favorable results upon decomposition presents a non-trivial challenge due to various factors. Firstly, determining the rank of a tensor is known to be NP-hard [111]. Additionally, while constructing a tensor that carries intuitive significance might be feasible, choosing the most suitable features for tensor dimensions and entries often demands thorough investigation. This process typically involves trial and error, experimenting with different feature combinations.

For instance, several previous studies utilized source IP, destination IP, and temporal information as tensor dimensions for network traffic data (Netflow). However, Netflow data encompasses numerous other potential features like bytes transferred, packet counts, source and destination port numbers, and connection durations, among others. Simultaneously, the choice of tensor entry could involve binary values, counts, or other Netflow features. Thus, the process of selecting the optimal feature combinations to define a tensor's dimensions and entries from a multitude of possibilities poses an exponential scale problem. With RFoT, we propose that employing an ensemble of randomized tensor configurations eliminates the necessity of identifying a single tensor with optimal dimensions and entry. Differently, with HNMFk Classifier and MalwareDNA, we make use of NMFk to heuristically estimate the rank, or the number of latent factors, of the tensor while the other features can be easily incorporated for analysis by unfolding the tensor among the first mode. Here the number of latent factors corresponds to the number of clusters. Heuristically estimating the ideal number of latent factors enable us to find the number of clusters that yield the accurate separation of the samples, and in return, finding clean signatures.

NMF has also been applied to the malware/benign-ware classification problem. Ling et al. derive similarity scores of structural patterns extracted with NMF to detect metamorphic malware (malware with the capability to modify its code during run-time) using static analysis features [118]. In their experiments, they choose a fixed number of components for NMF where the number of components k is selected as $k(n + m) < nm$. A single application of NMF misses the patterns hidden in malware sub-groups, and using a fixed number of components can result in missing important information (under-fitting) or including noise (over-fitting) in the results. Unlike Ling et al., we perform malware family classification by applying hierarchical NMF to discover the sub-groups and utilize NMFk as a heuristic to determine the number of components or clusters for HNMFk Classifier and MalwareDNA. Prior work outside the malware analysis field has demonstrated that hierarchical NMF can be used to achieve good clustering of the data [85, 166]. Gillis et al. show that using rank-two factorization at each step (i.e. split the data into two at each stage, $k = 2$) yields good clustering results when applied with hierarchical NMF [85]. We use hierarchical rank-two NMF in our ablation studies in Section 5.0.2, and show that estimating the number of components via *NMFk* produces better classification results, although extracting two clusters at each factorization does yield good classification results that surpass our baseline models. We attribute this performance gain to the advantage of using hierarchical approach that enable us to discover finer grained patterns hidden in the data.

The most analogous work to ours, concerning semi-supervised learning, was conducted by Irofti et al., who proposed a semi-supervised solution using Dictionary Learning (DL) for classifying Windows PE malware [96]. Their framework initially trains a dictionary in a supervised manner, facilitating the intermittent classification of new malware instances, subsequently updating the dictionary with signals from new malware in an unsupervised online fashion. However, DL is based on matrix factorization where the careful selection of number of latent features, or rank, is important to develop the best model as described in Section 2.0.5, which is not considered within this work while we utilize NMF with automatic model determination to carefully select the rank of the factorization. Specifically, we use NMFk for HNMFk Classifier and MalwareDNA. The careful selection of rank based on the heuristic in NMFk allow us to extract clean clusters which allow us to achieve improved downstream classification performance as compared to our baselines.

Chapter 4

Random Forest of Tensors

Tensor decomposition is as a powerful unsupervised machine learning technique capable of modeling multi-dimensional data, including that related to malware. This section discusses a method that employs tensor decomposition for malware analysis, or classification of malware and benign-ware. We introduce an innovative ensemble semi-supervised classification algorithm named Random Forest of Tensors (RFoT). RFoT leverages tensor decomposition to extract intricate latent patterns from the data. Our hybrid model combines multidimensional analysis with clustering to capture sample groupings within latent components, aiding in distinguishing between malware and benign-ware. The patterns extracted from malware data using tensor decomposition heavily rely on the configuration of the tensor, including dimension, entry, and rank selection. To encompass diverse perspectives offered by different tensor configurations, we adopt the 'wisdom of crowds' philosophy. This involves leveraging decisions made by the majority within a randomly generated ensemble of tensors, varying in dimensions, entries, and ranks. We illustrate RFoT's effectiveness in classifying Windows Portable Executable (PE) malware and benign-ware. To promote the utility of tensor decomposition for malware analysis and ensure the reproducibility of our results, we have made our code publicly available[1].

RFoT represents the first semi-supervised framework developed to address the challenges associated with large-scale malware analysis. This method demonstrates that tensor decomposition, within a hybrid framework combining clustering and ensemble learning, can effectively classify malware. Additionally, RFoT incorporates abstaining predictions, which help maintain its performance despite a reduced quantity of labeled data. We will now detail the RFoT methodology.

[1]RFoT is available at `https://github.com/MaksimEkin/RFoT`

4.0.1 Clustering Specimens Over the Latent Components

We find that malware and benign-ware samples can be separated in an unsupervised manner using tensor decomposition. This section begins by describing the necessary tensor configuration essential for extracting sample groupings from a tensor decomposition. Moreover, we present concrete instances of clusters representing groupings of malware and benign-ware within the latent factors. We then summarize the clustering methods used in RFoT to capture the patterns formed in the latent factors.

Malware Patterns in the Latent Factors

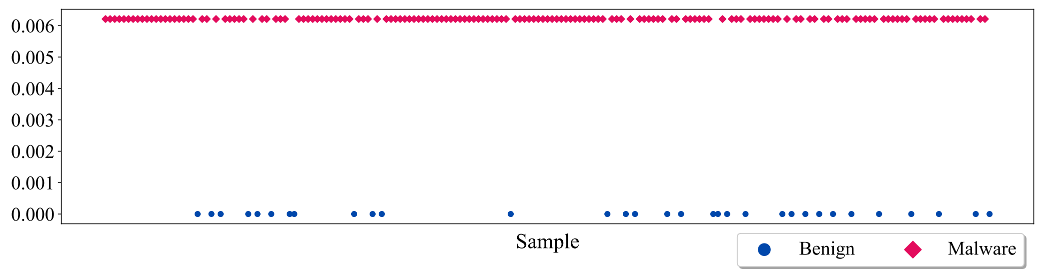


Figure 4.1: Clean malware and benign-ware clusters found by tensor decomposition

In order to extract latent factors with the capability of describing malware and benign-ware patterns based on each sample individually, we set our first

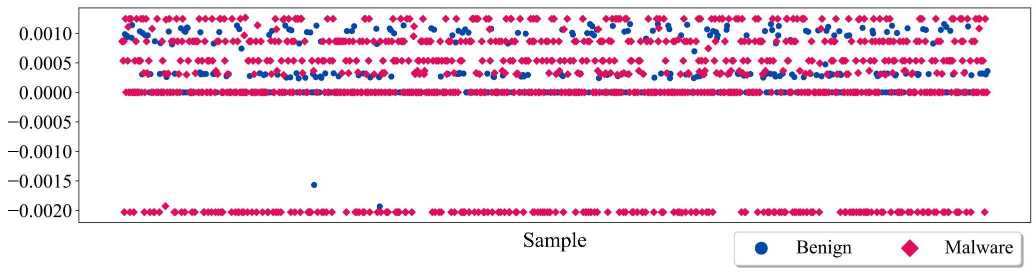


Figure 4.2: Clean malware clusters and noisy benign-ware clusters found by tensor decomposition

dimension of the tensor $\mathcal{X}$ to represent each malware sample while selecting the remaining of the $D-1$ dimensions and the tensor entry from static-malware-analysis based features using the PE header. We give a detailed description of how the remaining $D-1$ is configured to build the tensor from PE features in Section 4.0.2. In this tensor configuration, the shape of $\mathcal{X}$ is $N_1 \times N_2 \times \cdots \times N_D$, where N_1 is the total number of malware and benign-ware files from our dataset. For example, to access the tensor entry of the first specimen from the dataset, for features that are indexing at $i_2, \cdots, i_D$, we would index the tensor as $\mathcal{X}_{0,i_2,\cdots,i_D}$. Because the first dimension of $\mathcal{X}$ represents the samples, the obtained latent factor matrix for mode-1 is $\mathcal{M}_{::0} = \mathbf{A}^{(1)} \in \mathrm{I\!R}^{\mathrm{N_1 \, x \, R}}$, where R is the tensor rank and $\mathbf{A}^{(1)}$ carries latent information regarding the samples in our data. Using $\mathbf{A}^{(1)}$, we can access each individual latent factor $\mathbf{A}^{(1)}_{:r} \in \mathrm{I\!R}^{\mathrm{N_1 \, x \, 1}}$ in which the N_1 malware and benign-ware samples would form clusters. In Figure 4.1, we provide an example latent factor $\mathbf{A}^{(1)}_{:r}$

obtained by factorizing $N_1 = 10,000$ malware and benign-ware from the EMBER-2018 dataset using CP-ALS. Here, we can see that CP-ALS was able to cleanly separate malware and benign-ware instances within the latent factor. We also provide a second example with more noisy clusters in Figure 4.2. In Figure 4.2, around 7 lines forming clusters can be seen. Although the lines that have the majority of the samples from the malware class form cleaner clusters, there are other clusters where benign and malware samples are included within the same cluster. In Section 4.0.3, we will describe how we handle the more noisy clusters, or the clusters with poor uniformity where the majority of the cluster is not represented by a single class, using the *Cluster Uniformity Score*.

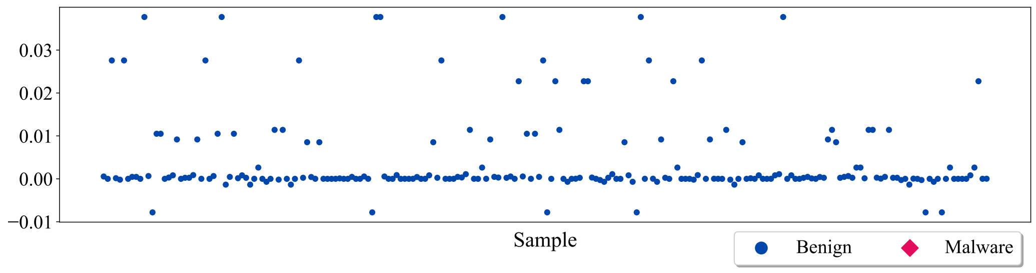


Figure 4.3: Tensor decomposition placing benign samples into a single latent factor

In addition to observing clusters that separate malware and benign-ware within each latent factor $\mathbf{A}^{(1)}_{:r}$, we also find that malware and benign instances cluster among components in $\mathbf{A}^{(1)}$, such that a single component r represents

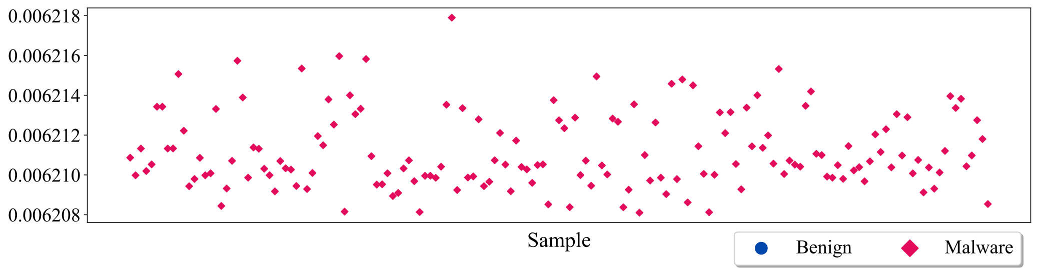


Figure 4.4: Tensor decomposition placing malware specimens into a single latent factor

samples from a single class. For instance, in Figure 4.3 we again show a latent factor obtained by factorizing $N_1 = 10,000$ malware and benign-ware from the EMBER-2018 dataset using CP-ALS. This time, it can be seen that CP-ALS was able to cluster benign instances within a single factor from the component r. Similarly, in Figure 4.4, it can be seen that the latent factor r only contains malware specimens. Motivated by the fact that we can acquire meaningful patterns that distinguish malware and benign-ware using tensor decomposition, we next look at how these patterns can be captured to enable building a semi-supervised classifier.

Capturing the Latent Patterns via Clustering

The performance of RFoT depends on the success in capturing the patterns found by tensor decomposition into clusters. Therefore, in this section we compare two different clustering methods. The first clustering algorithm we use to capture the patterns is called Mean Shift (MS) [84, 101, 175]. Specifically, we use the Scikit-learn implementation of this algorithm [139]. MS uses centroids to be the mean of clusters and updates the location of the clusters in a hill-climbing fashion to locate the maxima of a given density function, making it a good fit to perform clustering in a 1-dimensional space [139]. The window length, or the furthest point from the centroid of a cluster, is selected via the bandwidth. We use Scikit-learn's *estimate_bandwidth*[2] API to automatically determine the number of clusters. RFoT applies MS to each component within the latent factor for the first dimension $\mathbf{A}^{(1)}_{:r}$ from a given tensor decomposition. We extract a total of G_r clusters from each latent factor $\mathbf{A}^{(1)}_{:r}$, adding up to total of $G = G_0 + G_1 + \cdots + G_{R-1}$ clusters for a single tensor decomposition. We let the $\mathbf{g}_{j,r}$ represent a cluster with a set of samples from the rth component and jth cluster, where $0 \leq j \leq G_r - 1$.

In addition to the MS clustering, we use *Component* clustering. The motivation behind the Component clustering comes from our observation that we can obtain class-based groupings among components, rather than within individual latent factor, obtained from tensor decomposition. We discussed this in Section 4.0.1, where figures 4.3 (component with only benign sam-

[2]We heuristically set the *quantile* hyper-parameter to be 0.1 for estimating the bandwidth.

ples) and 4.4 (component with only malware specimens) showed an example of clean clustering within a single latent factor from the component r. Formally, when using the *Component* clustering, we will let each $\mathbf{A}^{(1)}_{:r}$ to define a single cluster such that $\mathbf{g}_{r,r} = \mathbf{A}^{(1)}_{:r}$. The total number of clusters from the component r in this case is $G_r = 1$, and the total number of clusters for the decomposition is $G = R$, where R is the tensor rank. Recall that figures 4.1 and 4.2 showed example latent factors where we had mix of both malware and benign-ware clusters. We will also use the *Cluster Uniformity Score*, introduced below at Section 4.0.3, to filter out the cases where we have more than one class describing a single latent factor.

After each tensor decomposition, we apply pre-processing to each latent factor $\mathbf{A}^{(1)}_{:r}$ to keep the samples with signals, or samples with a value that is not near 0 within the latent factor. To this end, prior to applying MS or Component clustering, we mask out (or remove) the points that are close to zero, where the distance to 0 is controlled with the hyper-parameter *zero_tol* in RFoT. In our experiments, we set $zero\_tol = 1e - 08$.

4.0.2 Ensemble of Random Tensor Configurations

Patterns extracted with tensor decomposition depends on the configuration of the tensor including the selection of the dimensions, tensor entry, and tensor rank. RFoT uses the *"wisdom of crowds"* philosophy by utilizing the patterns found from an ensemble of tensor configurations with randomly selected dimensions, entries, and ranks.

Notation for an Ensemble of Tensors

We use the variable *n_estimators* to represent the number of randomly generated tensor configurations. Let $\boldsymbol{\mathcal{X}}^{(i)}$ be one of the randomly generated tensors where i is in range $1 \leq i \leq$ *n_estimators*. To describe the random tensor configuration members of an ensemble, we re-formulate the notations introduced for tensor decomposition in Section 2.0.2. We begin with re-writing the CPD formula with sum of rank-1 tensors:

$$\boldsymbol{\mathcal{X}}^{(i)} \approx \sum_{r=1}^{R_i} \lambda_r \cdot \mathbf{a}_r^{(i,1)} \circ \mathbf{a}_r^{(i,2)} \circ \cdots \circ \mathbf{a}_r^{(i,D_i)} \tag{4.1}$$

Here we have the rank R_i CPD for the ith random tensor $\boldsymbol{\mathcal{X}}^{(i)}$ with D_i dimensions, and each $\mathbf{a}_r^{(i,d)}$ represents the rth latent factor for dimension d, where r is in range $1 \leq r \leq R_i$ and d is in range of $1 \leq d \leq D_i$. Following the KRUSKAL format we re-write the low-rank approximation as follows:

$$\mathcal{X}^{(i)} \approx \mathcal{M}^{(i)} \equiv [\![\lambda\ ;\ \mathbf{A}^{(i,1)}, \mathbf{A}^{(i,2)}, \cdots, \mathbf{A}^{(i,D_i)}]\!] \tag{4.2}$$

Here $\mathcal{M}^{(i)}$ is the low-rank estimation for the ith random tensor, and $\mathcal{M}^{(i)}_{::d-1} = \mathbf{A}^{(i,d)}$ is the latent factors matrix for dimension d. Each $\mathbf{A}^{(i,d)} \in \mathbb{R}^{\mathrm{N_d\,x\,R_i}}$ is a collection of latent factors as follows:

$$\mathbf{A}^{(i,d)} = [\mathbf{a}^{(i,d)}_1, \mathbf{a}^{(i,d)}_2, \ldots, \mathbf{a}^{(i,d)}_{R_i}] \tag{4.3}$$

As explained in Section 4.0.1, to capture the sample groupings, RFoT fixes the first dimension to represent each sample from our dataset. In an ensemble of random tensor configurations setting, $\mathcal{M}^{(i)}_{::0} = \mathbf{A}^{(i,1)} \in \mathbb{R}^{\mathrm{N_1\,x\,R_i}}$ is the latent factors matrix for the first dimension representing the N_1 malware and benign instances for the ith random tensor. MS clustering is applied to each $\mathbf{A}^{(i,1)}_{:r} = \mathbf{a}^{(i,1)}_r$, to capture $G^{(i)}_r$ number of clusters from component r, such that the total number of clusters found from ith tensor is $G^{(i)} = G^{(i)}_0 + G^{(i)}_1 + \cdots + G^{(i)}_{R_i-1}$. In an ensemble notation, we will let each cluster with a set of samples to be denoted with $\mathbf{g}^{(i)}_{j,r}$, where $0 \leq j \leq G^{(i)}_r$, for the rth component of ith tensor decomposition. With Component clustering, each cluster is $\mathbf{g}_{r,r}^{(i)} = \mathbf{A}_{:r}^{(i,1)}$, and the total number of clusters $G^{(i)} = R_i$ for the ith tensor decomposed to rank R_i.

Random Tensor Configuration Sampling

Our random tensor sampling includes a random selection of the number of dimensions, the features to represent each dimension, tensor entry, and random or fixed tensor rank. For each i random tensor, we first randomly choose the number of dimensions D_i with replacement, such that the range of D_i is $3 \leq D_i \leq \beta - 1$, where β is the total number of features from the original matrix $\mathbf{X} \in \mathbb{R}^{\mathrm{N_1 \, x} \, \beta}$. The minimum and maximum number of dimensions a random tensor configuration can have is controlled using the RFoT hyper-parameters (*min_dims*, *max_dims*), where *dims* is short for dimensions. Here *min_dims* $>= 3$ since tensors have at least 3 dimensions, and *max_dims* $<= \beta$ since we need one of the features to be the tensor entry. The first dimension is size N_1 representing each malware and benign-ware sample, and the features representing the remaining $D_i - 1$ dimensions are randomly selected from β features without replacement. Next, from the remaining $\beta - D_i$ features, which represent the feature(s) that are not selected to be a tensor dimension, RFoT randomly selects the feature to be used as the tensor entry with replacement. Finally, rank R_i is selected randomly, also with replacement, or each random tensor is assigned a user-defined fixed rank $R_i = rank$.

Tensor rank determines the number of hidden features, or latent components, that the tensor decomposition should extract. If we choose the rank to be too low, then we may miss vital information (*under-fitting*), while if the rank is chosen to be too high, then we might include noise in our solution (*over-fitting*) [169]. If we under-fit or over-fit the solution, then the latent factors might not have meaningful patterns to cluster benign and malware instances. By randomly selecting the rank, we attempt to avoid the need to correctly determine the rank of the tensor. Using the philosophy *"wisdom of crowds"*, we hope that an ensemble of tensor decomposition collectively can reach a consensus in the extracted patterns and allow precise malware detection. By doing so, we let the ensemble members, random tensors, complement each others' weaknesses. Specifically, we want to cancel out the impact of the cases where we obtain poor tensor decomposition results by deriving a decision based on the majority of the population. This hypothesis assumes that the cases with impure clusters do not represent the majority of the population.

As we sample each random tensor configuration, we do not check if the same configuration was already used. This check is avoided to ensure that the random sampling process remains fast as the size of the ensemble grows. Therefore, the aforementioned four steps for sampling random tensor configurations for building the ensemble of *n_estimators* tensors can result in repeated tensor configurations, which would need to be discarded after the sampling. Inspired from the previously introduced technique for fast sampling of zero tensor indices [92], we over-sample the tensor configurations to lower the probability of the repeated tensor configurations. Therefore, we set the ensemble size to be $n\_estimators + (n\_estimators \cdot 0.1)$ random tensors. We then perform post-processing to keep the unique tensor configurations and under-sample the ensemble to be the size of at most *n_estimators*.

Feature to Tensor Dimension Mapping and Tensor Entry

Categorical features can easily be mapped to an index in the tensor dimension. For example, take the EMBER-2018 feature *has_signature*, a binary feature that can be a 0 or 1. If a tensor dimension represents this feature, the size of that dimension would be 2, where $has\_signature = 0$ would map

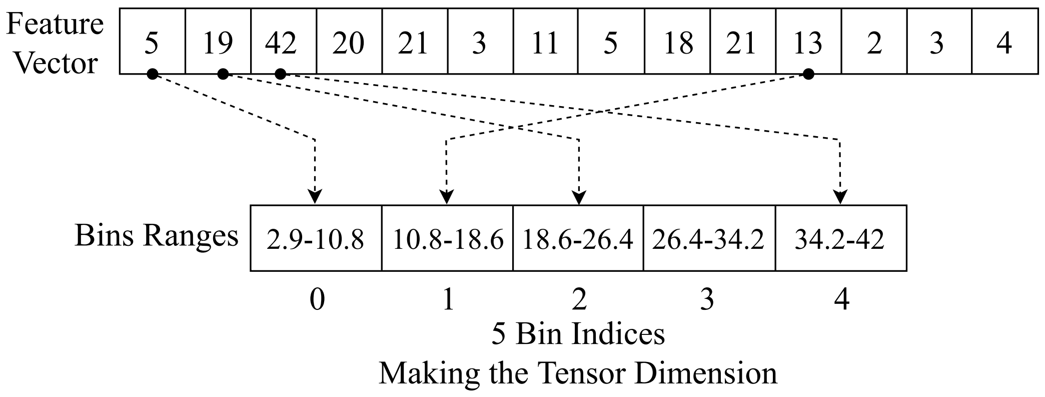


Figure 4.5: Example of 4 numerical feature values being mapped to 5 bins to form a tensor dimension.

to index 0, and *has_signature* = 1 would map to index 1. This generalizes to any categorical feature where the labels can be encoded to retrieve features to dimension index mapping.

On the other hand, in order to use a numerical feature as a tensor dimension we need to utilize binning to map the given numerical value to a certain index in the dimension. RFoT uses the *cut* API from the *Pandas* Python library to bin numerical values [137, 174]. The number of bins, or dimension size N_d, is determined by the RFoT hyper-parameter *bin_scale*, where the number of bins is $N_d = \mathit{bin\_scale} \cdot \text{num\_unique}(\mathbf{f})$. Here $\text{num\_unique}(\mathbf{f})$ gives the total number of unique elements present in a given feature vector $\mathbf{f}$, which is one of β features. We provide an example in Figure 4.5 which shows how numerical features are mapped to 5 bins to from a tensor dimension.

We can utilize a real example to further explain how a tensor for malware data can be built using numerical feature binning and categorical feature mapping. For example, let ith random tensor $\boldsymbol{\mathcal{X}}^{(i)}$ have the dimensions *Sample - Number of Strings - Has Signature*, and entry *Number of Sections*. *Sample* dimension represents each N_1 malware and benign-ware sample. The dimension *Number of Strings* represents the number of printable strings present in a given malware or benign instance, with size $N_2 = \textit{bin\_scale} \cdot \text{num\_unique}(\mathbf{f})$, such that the EMBER-2018 features for the number of strings will map to an index between 0 and $N_2 - 1$. The categorical dimension *Has Signature* identifies if a given specimen has a signature or not, thus the size of the last dimension is $N_3 = 2$. Finally, the tensor entry *Number of Sections* determines the number of sections present in the PE header of a given file. An entry $\boldsymbol{\mathcal{X}}^{(i)}_{n,s,f}$ in this tensor represents the number sections that a specimen $n \in [0, 1, \cdots, N_1 - 1]$, with number of strings that bins to an index $s \in [0, 1, \cdots, N_2 - 1]$, and with the signature flag $f \in [0, 1]$ has.

4.0.3 Semi-supervised Classification with RFoT

In this section, we describe the utilization of clusters for downstream classification within an ensemble framework. Additionally, we discuss the application of information gain to select the most relevant clusters for inclusion in the voting process.

Semi-supervised Voting Using the Clusters

Tensor decomposition extracts latent patterns from multi-dimensional data in an unsupervised fashion, and we capture these patterns for malware and benign samples using clustering techniques as described in Section 4.0.1. Using the captured clusters, we formulate a semi-supervised classifier that utilizes the information found by tensor decomposition. In this section, we first describe how the semi-supervised voting over the clusters is performed. This include the cases where the model is unable to make a decision for a given sample, and thus *abstaining vote* is given. We then introduce the *Cluster Uniformity Score* that is used as a threshold to filter out noisy, or non-uniform clusters.

RFoT takes a dataset $\mathbf{X} \in \mathbb{R}^{\mathrm{n\,x}\,\beta}$, where n is the number of samples and β is the number of features, and a vector $\mathbf{y}$ that represents the labels for

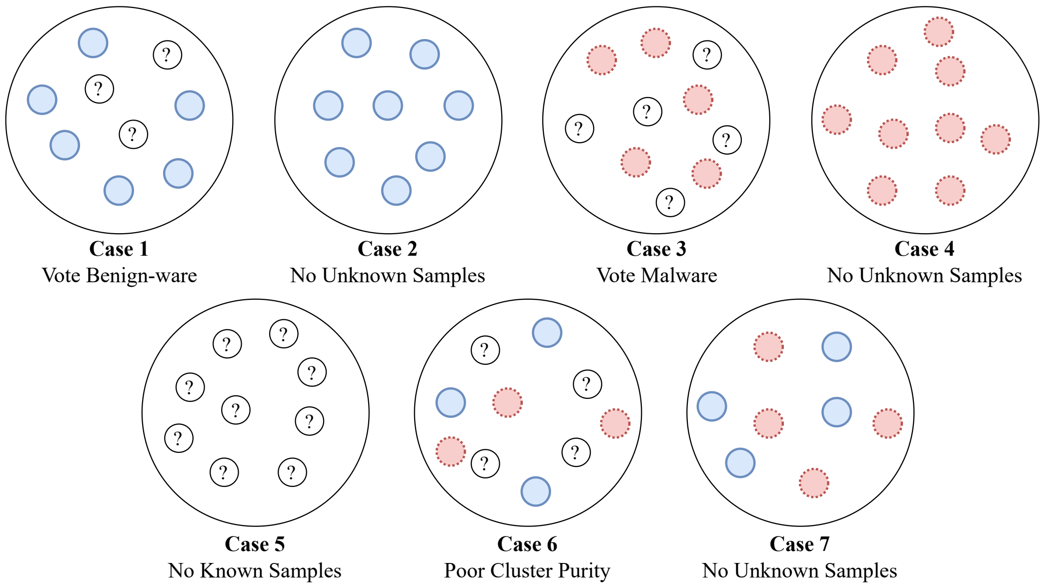


Figure 4.6: Possible cases of clusters that can be seen.

each n samples such that $y_n \in [-1, 0, 1, \cdots, C-1]$. Note that -1 is used for the unknown specimens, and C is the number of classes. In this chapter, we have $C = 2$ for malware and benign-ware, such that $y_n \in [-1, 0, 1]$ where 0 labels the benign-ware and 1 labels the malware. When we obtain a cluster, we use the known samples (samples with labels) as a reference to help us make a decision against the unknown samples (samples without labels, or -1) within that cluster. Specifically, the class vote of the given unknown samples that are in the cluster $\mathbf{g}_{j,r}^{(i)}$ is determined by the majority class of the known samples that are in the same cluster $\mathbf{g}_{j,r}^{(i)}$. There are 7 possible cases of cluster characteristics that we can obtain from the latent components, which are shown in Figure 4.6. In **Case 1**, we may have a cluster containing a set of unknown specimens and a set of known benign-ware. In this case, we would vote the unknown specimens as benign files. Similarly, we can vote the unknown specimens as malware if the majority of the known specimens are malware in the same cluster, as shown in **Case 3**. It is also possible to come across clusters where no unknown specimens are present, as shown in **Case 2**, **Case 4**, and **Case 7**. If there are no unknown samples in a given cluster, we disregard the cluster since we do not need to perform any voting.

This semi-supervised setup for classifying unknown specimens via clustering allows us to perform abstaining predictions (i.e. predict *"I do not know"*) due to not being able to obtain a class vote for a given sample. For instance, if a cluster consists of only a set of unknown specimens, as shown in **Case 5**, we cannot take a vote for these samples since we do not have any labeled instances to inform us regarding the class vote. We also cannot take a vote for the samples that are masked out due to the lack of signals (samples that are close to 0 with a certain threshold), as described in Section 4.0.1, since these instances would not be used in clustering. If a given sample always falls in a cluster without any known samples, as in **Case 5**, for each random tensor $\mathfrak{X}^{(i)}$ and its latent factor for the first dimension $\mathbf{A}^{(i,1)}$ obtained by the decomposition, then this sample is predicted to be abstaining, or its label is kept as unknown (-1). Similarly, if a given sample n is consistently masked out due to being near zero in each $\mathbf{A}^{(i,1)}$, then it is predicted to be abstaining.

For the samples that do get class vote(s), we perform max-vote to determine the final class prediction. That is, if a given specimen n has its majority of the votes (over 50%) representing one of the C classes, the instance n is predicted to be that class.

Cluster Uniformity Score

It is possible to encounter a cluster that is not uniform in representing a single class (cluster have known instances from multiple classes). We have already shown an example of a latent factor with non-uniform clusters in Figure 4.2, where noisy clusters occur. In Figure 4.6, **Case 6** demonstrates a cluster where we have a mix of known malware and benign-ware specimens. In such cases, we cannot obtain an accurate class vote from the cluster. To filter out these clusters, we use the *Cluster Uniformity Score* which is calculated based on the fraction of the most dominant known class in the given cluster $\mathbf{g}_{j,r}^{(i)}$. While we first begin using the cluster uniformity score for RFoT, this scoring technique is also used for HNMFk Classifier and MalwareDNA for determining the uniformity of the clusters based on known specimens. For RFoT, we utilize the same metric, but re-formulate it to match with our ensemble of tensors notation as follows:

$$U^{\mathbf{g}_{j,r}^{(i)}} = \frac{|\max(\mathbf{g}_{j,r}^{(i)^{known}})|}{|\mathbf{g}_{j,r}^{(i)^{known}}|} \tag{4.4}$$

Here $U^{\mathbf{g}_{j,r}^{(i)}}$ is the cluster uniformity score for jth cluster obtained from rth component of tensor decomposition of ith tensor, $\mathbf{g}_{j,r}^{(i)}$. $|\max(\mathbf{g}_{j,r}^{(i)^{known}})|$ is the number of samples that belongs to the most dominant class with known samples in the cluster $\mathbf{g}_{j,r}^{(i)}$, while $|\mathbf{g}_{j,r}^{(i)^{known}}|$ is the total number of known samples in the cluster. The clusters where $U^{\mathbf{g}_{j,r}^{(i)}}$ is below the specified uniformity threshold t are removed from consideration, and thus no class vote is obtained from these clusters. If a given specimen n continuously falls in the clusters that are removed due to poor purity, it is also predicted to be abstaining at the end.

4.0.4 Putting it Together: the RFoT Algorithm

We summarize the RFoT methodology in Figure 4.7, and with pseudocode in Algorithm 2. We first randomly sample tensor configurations (1).

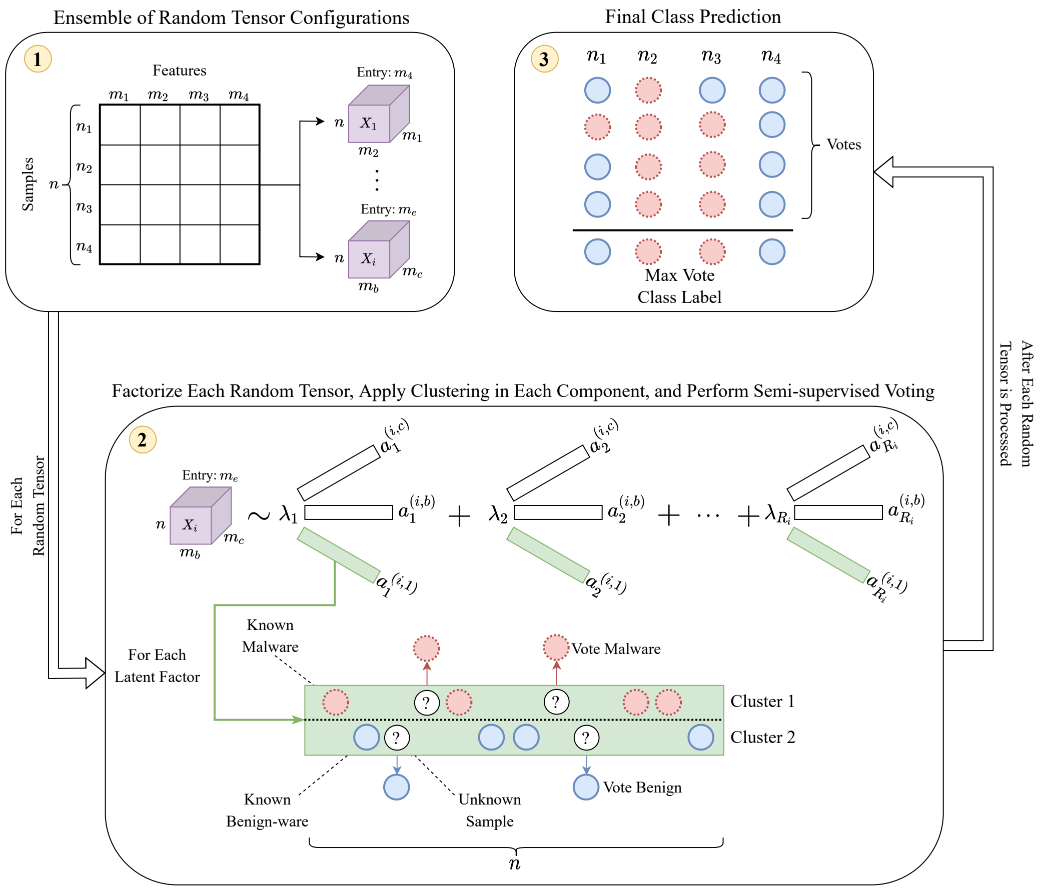


Figure 4.7: RFoT methodology overview

Then each tensor configuration is factorized to obtain the latent components (2). Within each latent component, we look at the latent factor representing each malware and benign-ware sample (2). Clustering is applied to capture the groupings within each of these latent factors (2). We filter out the noisy clusters using the cluster uniformity score. In the cases where we were able to acquire clean clusters, we take a class vote in a semi-supervised fashion (2).

After each tensor is factorized, and class votes are obtained from each latent factor for the first dimension, we get the final class prediction via max-vote (3). The specimens are predicted to be abstaining if they did not get any class vote due to either being part of clusters that were not uniform, not falling in a cluster that had known samples, or because they were masked out due to not having a signal.

Algorithm 2: RFoT($\mathbf{X}$, $\mathbf{y}$, *n_estimators*, *bin_scale*, t, R, *min_dims*, *max_dims*)

```
 1: tensor_configs = sample_tensors($\mathbf{X}$, $n\_estimators$, $R$, $min\_dims$, $max\_dims$)
 2: class_votes = []
 3: for config in tensor_configs do                          ▷ Start the parallel execution
 4:     $\mathcal{X}^{(i)}$ = build_tensor(bin_features($\mathbf{X}$, config), config)     ▷ COO format
 5:     $\mathcal{M}^{(i)}$ = decompose($\mathcal{X}^{(i)}$, $R_i$)                   ▷ CP-ALS or CP-APR
 6:     $\mathbf{A}^{(i,1)}$ = get_signals($\mathcal{M}^{(i)}_{::0}$)      ▷ Mask out near zero elements for the mode-1
 7:     clusters = cluster_latent_factor($\mathbf{A}^{(i,1)}_{:r}$)          ▷ For each $R_i$, MS or Component
 8:     for $\mathbf{g}^{(i)}_{j,r}$ in clusters do
 9:         if $\mathbf{g}^{(i)}_{j,r}$ in [Case 4,5,6 or 7] then      ▷ See Figure 4.6 for the cases
10:             continue                                        ▷ Abstaining votes
11:         else
12:             class_votes.append(vote($\mathbf{g}^{(i)}_{j,r}$, $\mathbf{y}$))      ▷ Semi-supervised voting
13:         end if
14:     end for
15: end for                                                ▷ End the parallel execution
        $\mathbf{y}_{\text{pred}}$ = max_vote(class_votes, $\mathbf{y}$)              ▷ Final class prediction
16: return $\mathbf{y}_{\text{pred}}$
```

We finally note that our implementation of RFoT computes the decomposition of the ensemble of random tensor configurations in a parallel fashion, since they are independent of one another. The parallel computation of the members of ensemble allows us to reduce the total time needed for prediction. Specifically, in Algorithm 2, lines 3 through 15 are executed in parallel based on the number of jobs that the user wants to run.

Now that we have introduced our methodology, we will next showcase the experiment results from a case-study where we classified malware and benign samples from the EMBER-2018 dataset using RFoT.

4.0.5 RFoT: Performance Analysis

In this section, we compare our results to the baseline models and assess the performance of RFoT alongside the baseline models as the labeled data percentage decreases.

Hyper-parameter Analysis

Prior to the comparison of our approach to the baseline models, we first evaluate its behaviour under different hyper-parameters including *Bin Scale*, *Cluster Uniformity Threshold*, *Max and Min Number of Dimensions*, *Tensor*

Rank, and *Number of Estimators*. The analysis into the hyper-parameter settings allow us to learn the best model setting for the dataset that we use, and also to understand the limitations and capabilities of RFoT. We conduct the hyper-parameter analysis using the CP-ALS tensor decomposition algorithm, and MS and Component clustering methods. For CP-ALS, we set the maximum number iteration to be 250 for each experiment.

Bin Scale

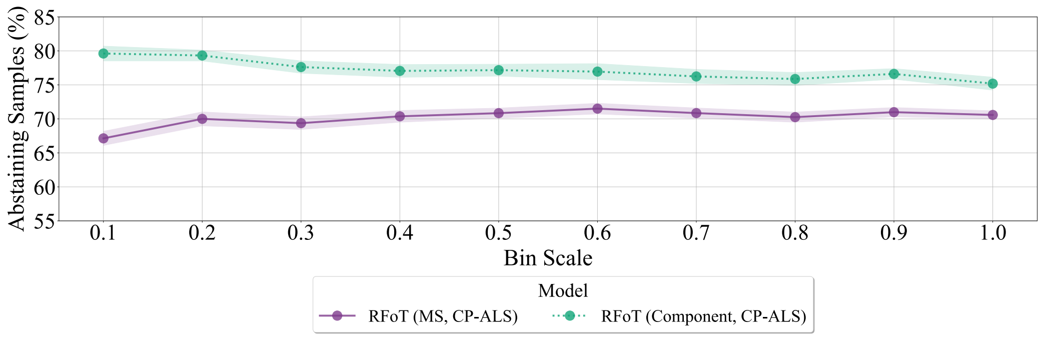


Figure 4.8: Abstaining percentage is shown for different values of the Bin Scale.

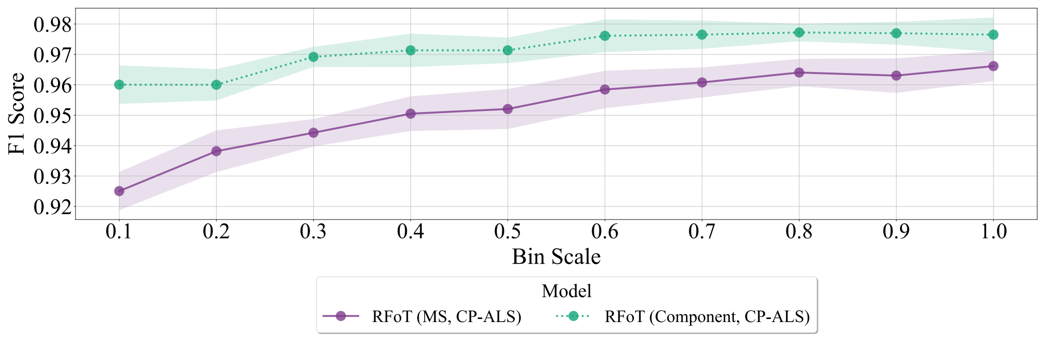


Figure 4.9: F1 score is shown for different values of the Bin Scale.

The first hyper-parameter that we investigate is *Bin Scale*, which determines the size of the tensor dimensions that represent the features with numerical values, as described in Section 4.0.2. During this analysis, we perform prediction while changing the *bin_scale* hyper-parameter between 0.1 and 1.0 with the step-size of 0.1. We set the remaining of the hyper-parameters as follows: *min_dimensions* = 7, *max_dimensions* = 8, *cluster_purity_tol* = 1.0, *rank* = 2, and *n_estimators* = 1000. Here *min_dimensions* and *max_dimensions* determines the minimum and the maximum number of dimensions any random tensor can have respectively. Since this experiment evaluates the bin scale hyper-parameter which only effects the features with numerical values, to ensure that each random tensor would have a dimension representing a feature with a numerical value we set the minimum number of dimensions to be 7. *cluster_uniformity_tol* gives us the threshold to select any given cluster to be used in voting, as described in Section 4.0.3. Using the hyper-parameter *rank*, we set each random tensor to be decomposed with rank-2. Finally, *n_estimators* determines the number of random tensors to be used.

In Figure 4.8, we can see that as the bin scale is increased, the percent of abstaining predictions for RFoT with Component clustering drops from around 80% to 75% while it increases from around 66% to 70% for MS clustering. At the same time, Figure 4.9 shows that the F1 score for RFoT with both MS and Component clustering increases as the bin scale reaches 1.0. When the bin scale is set to 1.0, the size of the dimension representing the given feature is equal to the number of unique values present in the given feature vector $\mathbf{X}_{:\mathbf{f}}$. The results in this analysis suggest that reducing the dimension size (or the number of bins) for the given numerical features, could result in under-fitting, or missing the details that help separate the malware and benign-ware. In addition, the fact that RFoT with Component clustering saw both reductions on abstaining prediction while increasing F1 score further supports RFoT being able to make a more precise decision for the given labels when we use a higher bin scale value.

Notice that RFoT with Component clustering yields better malware-detection results, but higher abstaining predictions. This could be attributed to MS clustering being able to extract meaningful clustering results that separate malware and malware from a single component, while the Component clustering misses the cases such as the one shown in Figure 4.1. Finally, we do see a larger performance improvement for the MS clustering, which indicates that the added information with the increased dimension size could be resulting in cleaner in-component clusters that separate classes.

Cluster Uniformity Threshold:

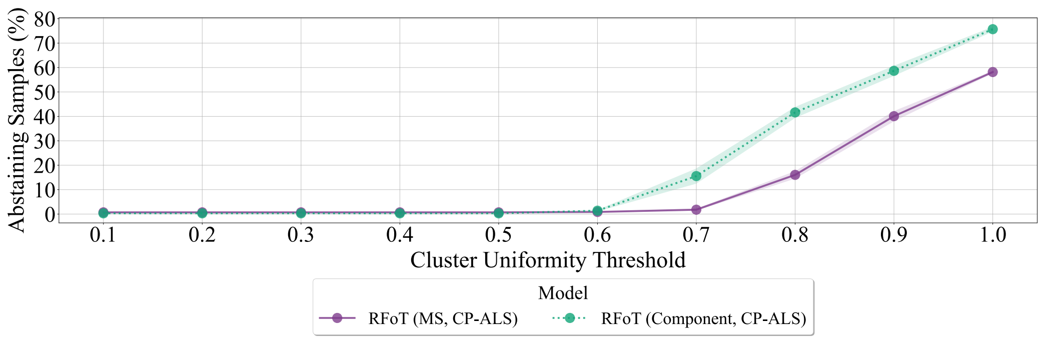


Figure 4.10: Abstaining percentage is shown for different values of the Cluster Uniformity Threshold.

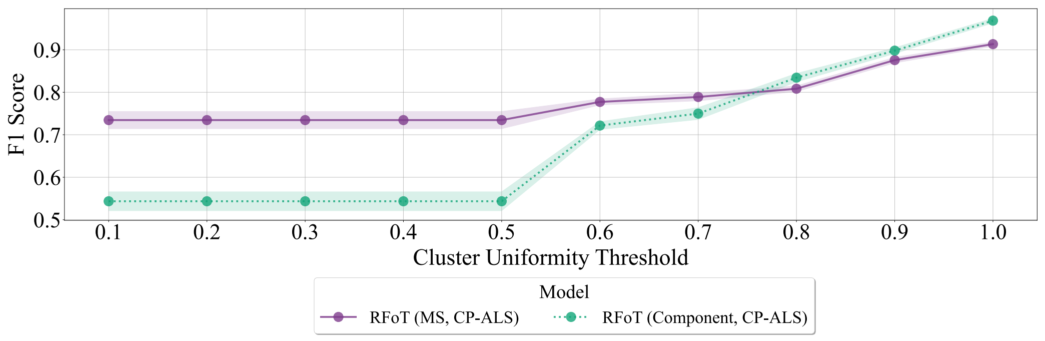


Figure 4.11: F1 score is shown for different values of the Cluster Uniformity Threshold.

We next look at the cluster uniformity threshold, which determines the threshold to remove the noisy clusters. In this analysis, the uniformity threshold is varied between 0.1 and 1.0 with a step-size of 0.1. The remaining of the hyper-parameters are kept same as the Bin Scale experiment form Section 4.0.5 except the following: $min\_dimensions = 3$ and $bin\_scale = 1$.

The first point to note in the results is that we do not get any abstaining predictions until after the uniformity threshold of 0.6 as shown in Figure 4.10. This results in poor performance for both MS and Component-based RFoT as shown in Figure 4.11 with low F1 scores. It can be seen, however, the F1 score improves rapidly as the cluster uniformity score is increased. The increase in the threshold also increases the abstaining predictions. This occurs because when we choose a higher cluster purity threshold, the clusters to be used in voting need to be cleaner such that the known classes in the cluster should represent mainly a single class. Specifically, it there should only be a single known class in the cluster when $cluster\_uniformity\_tol = 1$. When we encounter clusters with known samples from a mix of different classes, using the threshold, the clusters are removed from the consideration for voting. This describes the reason behind the increased abstaining predictions. Since we begin to use only the cleaner clusters, we do see the increased performance of the model.

The final point to note in this experiment is that MS-based clustering does outperform Component-based clustering with lower values of cluster uniformity threshold. This could indicate that the CP-ALS algorithm is more capable of finding meaningful in-component clusters separating classes rather than among-component clusters where each component individually separates classes.

Maximum and Minimum Number of Dimensions:

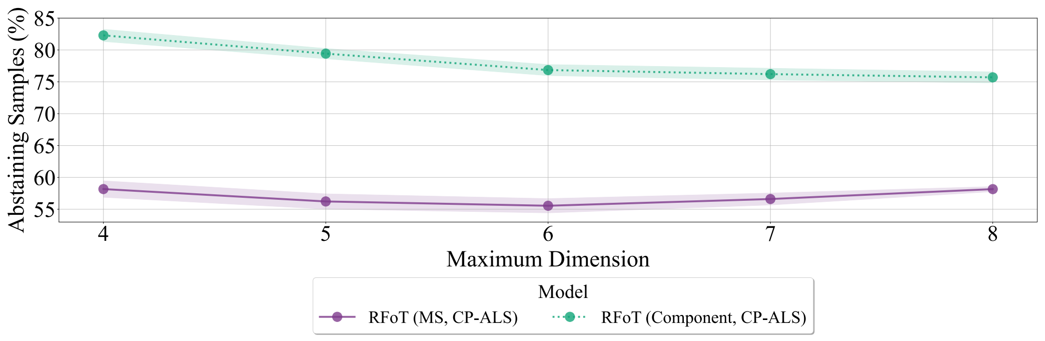


Figure 4.12: Abstaining percentage is shown for different values of the Max Dimensions parameter.

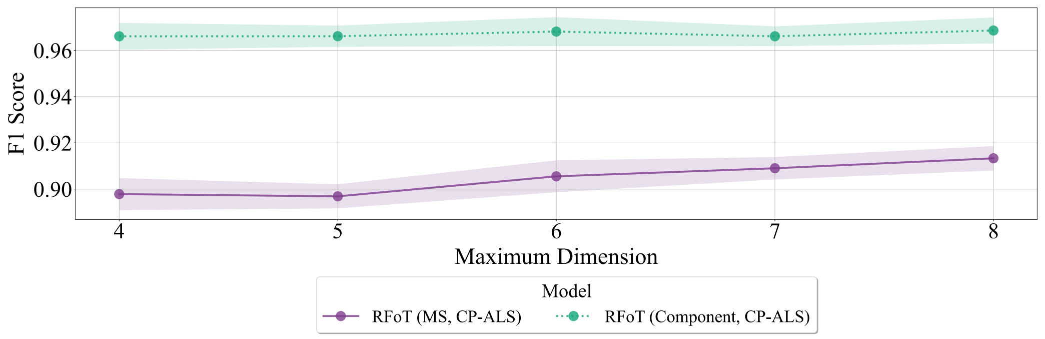


Figure 4.13: F1 score is shown for different values of the Max Dimensions parameter.

The next hyper-parameters we investigate are the choice of the minimum and the maximum number of dimensions the random tensors should have within the ensemble. To test this, we set the maximum number of dimensions between 4 and 8, with a step size of 1. For the minimum number of

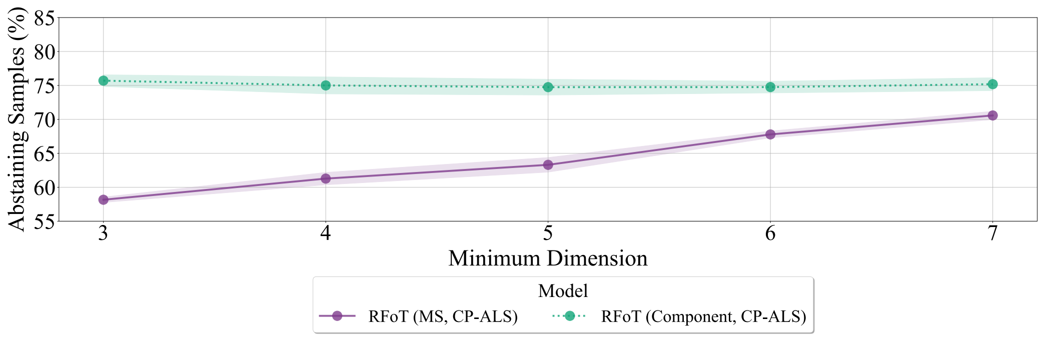


Figure 4.14: Abstaining percentage is shown for different values of the Min Dimensions parameter.

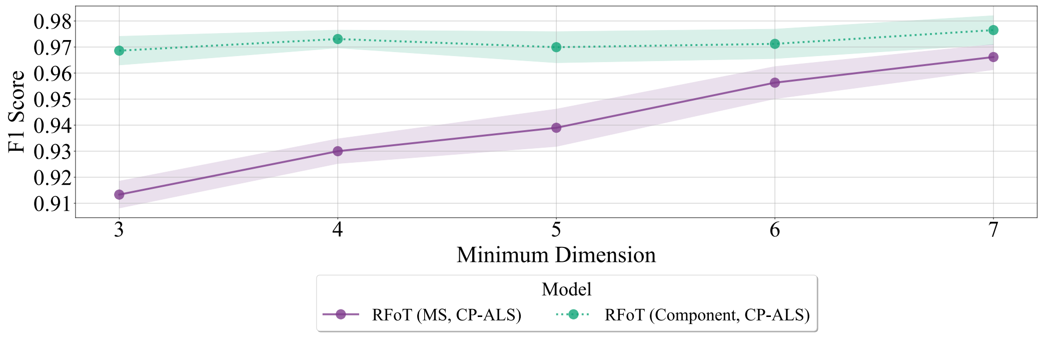


Figure 4.15: F1 score is shown for different values of the Min Dimensions parameter.

dimensions, we look at between 3 and 7 with a step size of 1. The remaining hyper-parameters are kept the same as the previous experiment in Section 4.0.5. Figures 4.12 and 4.13 show that we get relatively stable results for increasing maximum number of dimensions. We observe a similar trend for the Component clustering-based RFoT as the minimum number of dimensions hyper-parameter is increased, as shown in Figures 4.14 and 4.15. However, we do see an improvement in malware classification for the MS clustering-based RFoT as the minimum number of possible dimensions is increased as shown in Figure 4.15. At the same time, there is an increase in the abstaining predictions for RFoT with MS clustering as shown in Figure 4.14. This result could indicate that CP-ALS begins to extract increased number of latent factors with noisy patterns; however, the patterns that do give meaningful result are more uniform.

Tensor Rank:

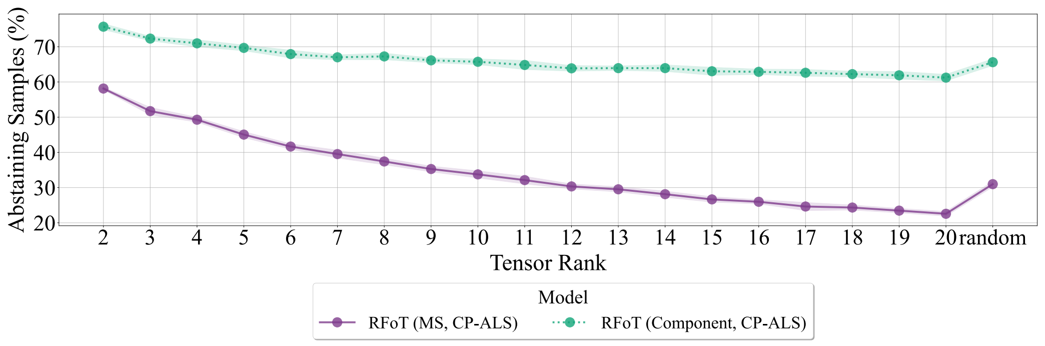


Figure 4.16: Abstaining percentage is shown for different values of the Rank parameter.

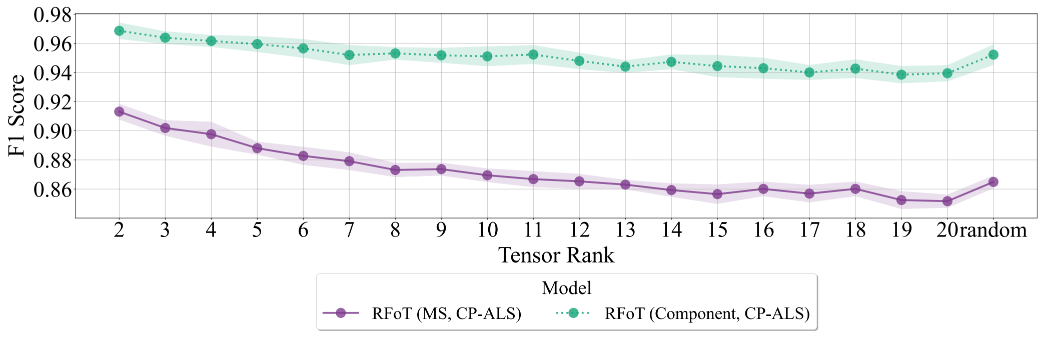


Figure 4.17: F1 score is shown for different values of the Rank parameter.

In our random ensemble of tensor configurations model, one of the tensor settings that can be randomly sampled is the tensor rank. We next look at the performance of RFoT with increasing fixed rank (where each tensor in the ensemble is decomposed with the same rank), and also with the randomly selected rank. The goal of this analysis to determine if the patterns extracted from an ensemble of tensor decompositions with randomly selected ranks can reach to a consensus in determining the class of the given samples, while avoiding the need to correctly determining the rank. This hypothesis also aligns with the need for the cluster uniformity calculations, where if the extracted patterns result in noisy clusters due to over-fitting or under-fitting, then we would want to remove these clusters using the cluster uniformity threshold.

For the fixed ranks, we test RFoT for MS and Component clustering where the ranks are ranged between 2 and 20 with the step size of 1. We also randomly choose the rank in this experiment for comparison to the fixed rank. The remaining of the hyper-parameters are as follows: $\textit{min\_dimensions} = 3$, $\textit{max\_dimensions} = 8$, $\textit{bin\_scale} = 1.0$, $\textit{cluster\_purity\_tol} = 1.0$, and $\textit{n\_estimators} = 1000$.

In Figure 4.16, we can see that as the rank is increased the percent of abstaining predictions for both RFoT with Component and MS clustering drops. This drop is more significant for the MS clustering based RFoT. As the number of components, or the rank, is increased, RFoT obtains more opportunities to find clusters to retrieve class votes. Increase in the total number of clusters, which results in increased number of possible class votes for the unknown specimens could describe the reason behind the drop in the number of abstaining predictions. At the same time, the steeper decline in the number of abstaining predictions for RFoT with MS clustering indicates that the increasing rank improves the capability of CP-ALS to extract patterns where in-component groupings separate malware and benign-ware.

As the abstaining predictions drop with the increasing rank, the performance of the model slightly drops as shown in the Figure 4.17. We again see a higher drop, compared to the Component clustering, for the MS clustering which could be the result in the steep decline in the number of abstaining predictions. Finally, we see that random selection of rank plays a role to smooth out the results for both the F1 score and the number of abstaining predictions.

Number of Estimators (Random Tensors):

The final hyper-parameter that we investigate is the selection of the total

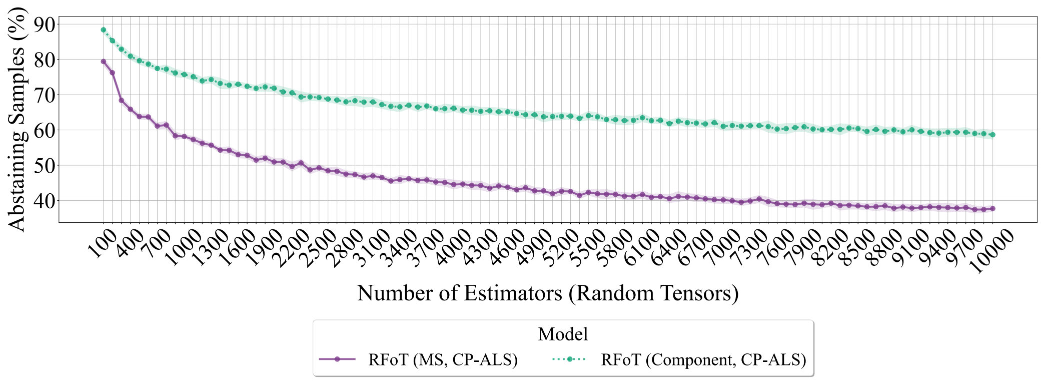


Figure 4.18: Abstaining percentage is shown for different values of the Num Estimators parameter.

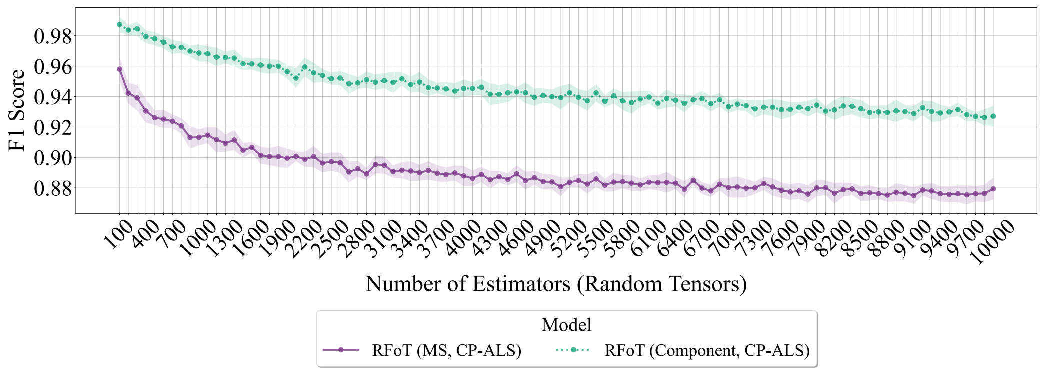


Figure 4.19: F1 score is shown for different values of the Num Estimators parameter.

number of estimators, or number of random tensor configurations in our ensemble. We test number of estimators between 100 and 10,000 with the step size of 100. The remaining of the hyper-parameters are kept as same as the ones we used in Section 4.0.5, except we use a fixed rank of $rank = 2$. We select a low-rank of 2 motivated from the results presented in Section 4.0.5. Although we might under-fit the data, cluster purity score allows us to separate out the noisy results and only keep the meaningful patterns, which enables achieving a better malware prediction accuracy.

We observe that as the number of estimators increases, the number of abstaining predictions drops as shown in Figure 4.18. The increased number of estimators also increases the number of clusters obtained from each tensor decomposition where we can potentially obtain class votes. The dropping number of abstaining predictions with the increasing number of estimators could be due to the increase in the votes. With the dropping number of abstaining predictions, we also see a decline in F1 score as shown in Figure 4.19. However, for both the F1 score and the number of abstaining predictions, we see that the decline begins to flatten. Therefore, although the increased number of votes results in a drop in performance, as we begin to obtain more votes from a larger ensemble, the model begins to make better decisions and slow down the performance decline. The performance of our model converging to a plateau shows that RFoT is capable of using the decision made from a majority of random tensors to obtain accurate predictions when the size of our ensemble is large enough. Differently, when the ensemble is small, RFoT is more certain (accurate) for the predictions made, but a smaller number of samples are predicted due to the high abstaining prediction percentage.

Baseline Comparisons

Model	Method	F1	Precision	Recall	Abstaining (%)	Time (sec)
RFoT (Component, CP-ALS)	Semi-supervised	**0.968** (+-0.005)	**0.968** (+-0.005)	**0.968** (+-0.006)	75.703 (+- 0.863)	536.151 (+- 5.132)
RFoT (MS, CP-ALS)	Semi-supervised	0.913 (+-0.005)	0.915 (+-0.004)	0.913 (+-0.005)	58.158 (+- 0.399)	554.831 (+- 5.263)
RFoT (Component, CP-APR)	Semi-supervised	0.940 (+-0.016)	0.941 (+-0.016)	0.940 (+-0.016)	93.220 (+- 1.534)	880.700 (+- 21.192)
RFoT (MS, CP-APR)	Semi-supervised	0.793 (+-0.008)	0.805 (+-0.007)	0.797 (+-0.008)	54.218 (+- 1.646)	1582.156 (+- 20.056)
LightGBM	Supervised	0.871 (+-0.005)	0.871 (+-0.005)	0.871 (+-0.005)	NA	**78.595** (+- 6.040)
XGBoost	Supervised	0.873 (+-0.005)	0.874 (+-0.005)	0.873 (+-0.006)	NA	93.805 (+- 2.752)
XGBoost+SelfTrain	Semi-supervised	0.872 (+-0.006)	0.873 (+-0.006)	0.873 (+-0.006)	NA	87.410 (+- 6.813)

Table 4.1: Baseline comparisons

We compare RFoT with CP-ALS and CP-APR decomposition, using MS and Component clustering, against baseline models *XGBoost*, *LightGBM*, and *XGBoost+SelfTrain*. For CP-APR we use 16 parallel jobs to decompose each random tensor using GPUs, while for CP-ALS is decomposed with 50 parallel jobs on CPUs. We tune the baseline models using a popular Python package *Optuna* [9]. *XGBoost* and *LightGBM* tuned with 3-fold stratified cross-validation and 50 trials to identify the optimal hyper-parameters. The tuning settings, or search space for the optimal hyper-parameters, listed below are the same from our experiments with HNMFk Classifier and MalwareDNA for semi-supervised malware family classification [59].

For *LightGBM*, we used 250 maximum number of iterations, *gbdt* boosting type, and objective function *binary_logloss*. The following hyper-parameters were tuned (ranges are shown in parenthesis): *min_data_in_leaf* (5-100 in log scale), *max_depth* (2-7), *bagging_freq* (0-5), *bagging_fraction* (.5-1.0), *learn-*

ing_rate (.001-.1 in log scale), and *feature_fraction* (.1-.7).

As for *XGBoost*, we set the maximum boosting rounds to 250 and use the *binary-hinge* objective function. The following hyper-parameters were tuned, with ranges again shown in parentheses: *max_depth* (2-10), *eta* (.003-0.5 in log scale), *subsample* (.2-.7), *rounds* (10-300), *colsample_bytree* (.3-1.0), *colsample_bylevel* (.5-1.0), and *lambda* (.1-2.0). We use the same tuned hyper-parameters for *XGBoost* for the *XGBoost+SelfTrain* baseline model.

In Table 4.1, we compare RFoT with baseline models based on F1 score, Precision, Recall, and computation time in seconds. Additionally, we display the percentage of abstaining predictions for RFoT. From the table, it's evident that each RFoT model outperforms every other baseline model. The RFoT model employing Component clustering and CP-ALS tensor decomposition achieves the highest F1 score of 0.968. However, this model also generates a significantly high abstaining prediction rate of 75.70%. Thus, an ideal model choice emerges with RFoT utilizing MS clustering and CP-ALS tensor decomposition, boasting a commendable F1 score of 0.91 and a lower abstaining prediction rate of 58%.

CP-APR with component clustering also demonstrates high performance with an F1 score of 0.94. However, it registers the highest abstaining prediction rate at 93.22%. In contrast, RFoT with CP-APR decomposition and MS clustering yields the lowest F1 score of 0.79. Among the models compared, the fastest performer is *LightGBM* clocking in at 78.59 seconds.

These results underscore that RFoT is an ideal model for precise malware detection, although it might predict a lower number of samples due to abstaining predictions. Notably, as a semi-supervised solution, RFoT surpasses supervised models, potentially offering better generalizability to novel malware.

Labeled Malware Data Scarcity Experiment

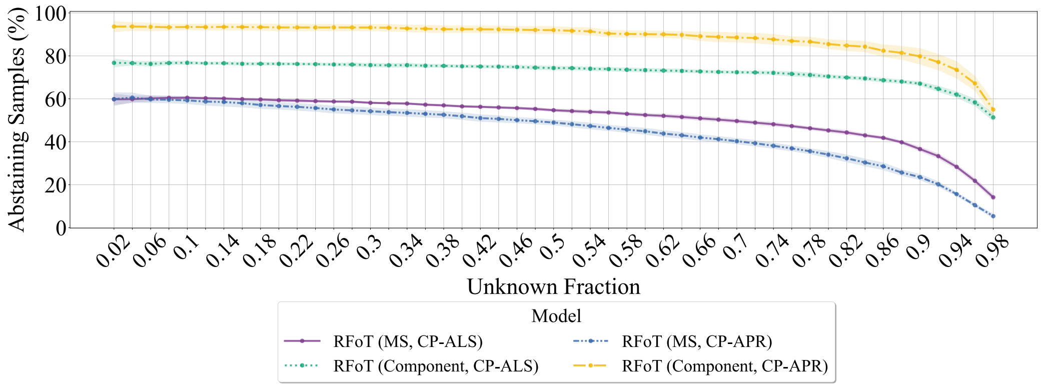


Figure 4.20: Abstaining percentage is shown for different values of the unknown fraction.

Compared to other ML fields, obtaining labeled malware data is time-consuming and expensive [148]. This issue is particularly problematic be-

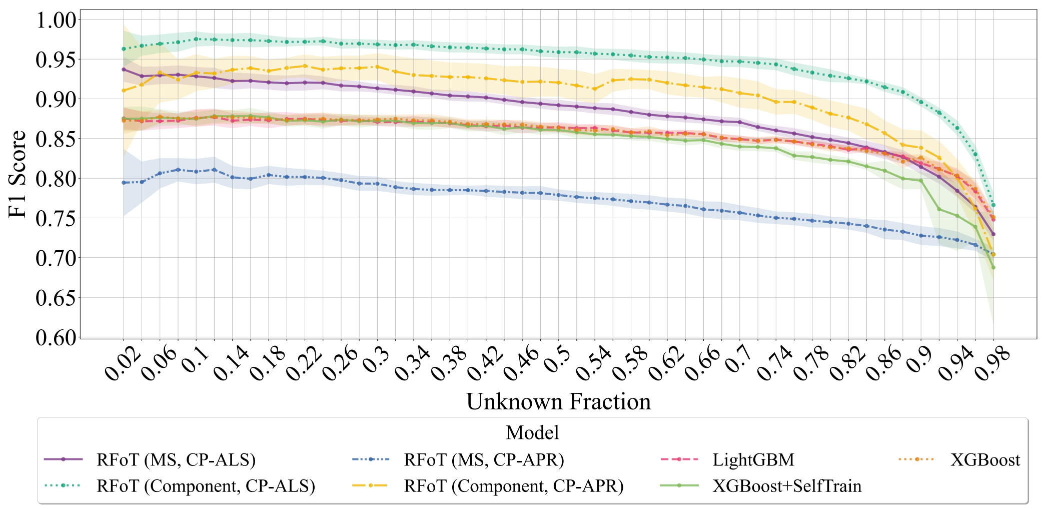


Figure 4.21: F1 score is shown for different values of the unknown fraction.

cause popular supervised ML solutions for malware detection often require a large quantity of labeled data to achieve good performance. Additionally, Raff et al. emphasize that semi-supervised solutions in the realm of Windows malware classification have not received sufficient attention, despite their potential benefits such as improved generalizability to novel malware and achieving higher performance even with a limited quantity of labeled data [148].

Therefore, we conducted tests on the performance of our semi-supervised solution with a decreasing quantity of labeled data. We then compared its performance with that of the supervised and semi-supervised baseline models.

We range the fraction of unknown specimens between 0.02 and 0.98 with the step size of 0.02. The fraction of unknown samples θ means that the proportion of the known samples would be $1-\theta$. For the supervised baseline models, the fraction of the unknown samples θ is equivalent to the size of the test set, while the fraction of the known samples determines the training set size. The baseline models are also tuned in this experiment.

In Figure 4.20, the percentage of abstaining predictions for RFoT with CP-ALS and CP-APR, along with MS and Component clustering, is illustrated. CP-APR with Component clustering exhibits the highest number of abstaining predictions, while CP-APR with MS clustering shows the lowest. Meanwhile, Figure 4.21 indicates that CP-APR with MS clustering demonstrates the lowest performance. This suggests that CP-APR struggles to identify meaningful patterns that differentiate the classes within each latent factor. However, CP-APR with Component clustering achieves high-performance results, albeit with a trade-off of a high number of abstaining predictions.

CP-ALS with MS and Component clustering presents a lower percentage of abstaining predictions along with higher F1 scores. As the fraction of unknown specimens increases, the percentage of abstaining predictions initially remains stable and then rapidly decreases, particularly after an unknown fraction of approximately 0.7, as depicted in Figure 4.20. This decline could be attributed to the cluster uniformity score's inability to filter out noisy clusters, as they are now represented by unknown specimens from the same class. Consequently, the other known specimens, which initially revealed the poor uniformity, are lost as the unknown fraction increases.

In Figure 4.20, it is noticeable that our baseline models generally demonstrate similar performance trends as the fraction of unknown specimens increases. However, there's a significant performance drop observed for *XGBoost+SleftTrain* after an unknown fraction of 0.74.

RFoT with CP-ALS and MS clustering, along with CP-APR with Component clustering, outperforms each baseline model until *XGBoost* and *LightGBM* start to surpass RFoT with CP-ALS and MS clustering after the unknown fraction reaches 0.86. Similarly, *XGBoost* and *LightGBM* outperform RFoT with CP-APR and Component clustering after an unknown fraction of 0.94.

Considering that abstaining predictions contribute to maintaining model performance, it is noteworthy that RFoT based on CP-ALS with Component clustering consistently outperforms each of the baseline models, irrespective of the fraction of unknown specimens.

We demonstrated the performance of RFoT and compared it to the tuned baseline models that prior studies have used to report state-of-the-art malware detection results. Our findings showed that RFoT, as a semi-supervised solution, exhibits superior capabilities in detecting malware compared to the baseline models, including the supervised and semi-supervised algorithms with a trade-off in lower coverage rate. Furthermore, our experiments highlighted that RFoT can achieve higher accuracy in detecting malware even as the percentage of known samples decreases.

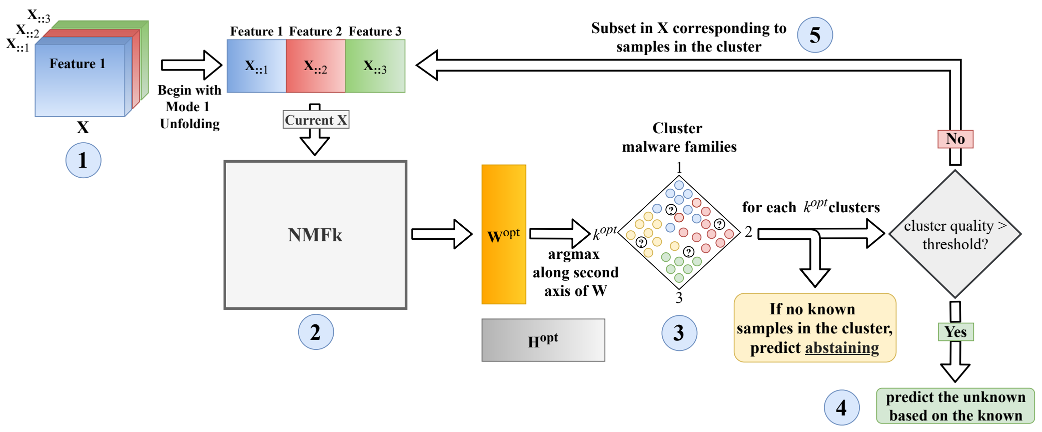
Chapter 5

HNMFk Classifier

The HNMFk Classifier is the second framework we have developed to address the shortcomings of large-scale malware analysis. While the RFoT, a tensor decomposition-based semi-supervised framework, showed promising results for malware classification, it lacked the capability to detect novel malware families and to perform effectively under class-imbalance conditions. With the HNMFk Classifier, we build on our experience from RFoT and address these critical issues by enhancing our model's ability to identify novel malware and operate under class imbalance. Further, while with RFoT we focused on malware detection, with HNMFk Classifier we adress a different problem of malware family classification.

The experiments conducted with the HNMFk Classifier follow the most realistic setup for our analysis and have set a world record [134]. This advanced framework not only improves upon the foundational aspects introduced by RFoT but also extends its applicability and efficacy in dynamic real-world scenarios.

5.0.1 HNMFk Classifier Algorithm



X::3
X::2
X::1
Feature 1
X
1
Begin with Mode 1 Unfolding
Feature 1 Feature 2 Feature 3
X::1
X::2
X::3
Current X
Subset in X corresponding to samples in the cluster
5
No
NMFk
2
W^opt
H^opt
k^opt
argmax along second axis of W
Cluster malware families
1
2
3
for each k^opt clusters
If no known samples in the cluster, predict abstaining
cluster quality > threshold?
Yes
4
predict the unknown based on the known

Figure 5.1: Overview of the HNMFk Classifier framework. NMFk is wrapped around an hierarchical (or recursive) semi-supervised architecture. Begin with the initial data **X** (1). Use NMFk to estimate the number of clusters and obtain the latent factor **W** (2). Extract the clusters via argmax along the second axis of **W** (3). For each cluster, perform abstaining prediction if no known samples are present in the cluster, or predict the unknown specimens in a semi-supervised manner if the cluster uniformity score is satisfied (4). Form the new matrices **X** with the specimens from the clusters that does not meet the cluster uniformity threshold (5). For each new **X**, apply NMFk again (2).

In this section, we describe how our NMFk based hierarchical bulk classifier, named HNMFk Classifier, works. HNMFk Classifier implements some of the ideas of MalwareDNA, without the real-time classification capability of MalwareDNA. Specifically, HNMFk Classifier directly works on the extracted clusters rather than utilizing a signature archive $\mathbf{M}$, which we introduce later in the dissertation. HNMFk Classifier performs bulk classification where the known samples are used as a reference against the unknown specimens when performing hierarchical clustering, resulting in a model with only an inference process (i.e. no training). Therefore, in comparison to the traditional ML models which have separate training (slow) and prediction (fast) steps, this solution can be used outside the real-time environments, such as early stages in the labeling process of the malware. Our model recursively analyzes the known and unknown specimens, factorizing only the subset of data from the previous cluster at each iteration.

The hyper-parameters of our model are the hyper-parameters needed for NMFk, and the cluster uniformity threshold t. The user specifies the maximum number of iterations for NMF, number of perturbations, the error rate, and the range to search for the k heuristic. When performing classification, *HNMFk Classifier* is provided with the data matrix $\mathbf{X} \in \mathbb{R}_{+}^{n \times m}$, where n is the number of malware samples and m is the number of features, which includes both the known and unknown specimens that we want to perform inference on. We also provide a vector $\mathbf{y}$ containing the labels for each specimen. The ith sample, where $1 \geq i \geq n$, has the family label $\mathbf{y}_i \in \{-1, 1, 2, \ldots, C\}$ for a dataset with C classes. Notice that the unknown specimens are labeled with -1.

Our algorithm proceeds with the first factorization, given $\mathbf{X}$, $\mathbf{y}$, the specified NMFk hyper-parameters, and cluster uniformity threshold t as input. After NMFk identifies the number of clusters k^{opt}, we obtain the latent factors $\mathbf{W} \in \mathbb{R}_{+}^{n \times k^{opt}}$ and $\mathbf{H} \in \mathbb{R}_{+}^{k^{opt} \times m}$. *HNMFk Classifier* uses $\mathbf{W}$ latent factor to perform clustering, which we call *W-clustering.* Here each n sample is assigned to one of k^{opt} clusters by taking the maximum value along the second axis:

$$\text{cluster}(i) = \underset{0 \leq j \leq k^{opt}}{\arg\max} \left(\mathbf{W}_{ij}\right) \tag{5.1}$$

where cluster(i) returns the cluster assignment of a given sample i. If a cluster c, where $c \in \{1, 2, \ldots, k^{opt}\}$, does not contain any known samples, all the unknown specimens in the cluster c are predicted abstaining. If a cluster c only has known specimens, we do not proceed with the samples in that cluster further, as there are no more unknown specimens to label. On the other hand, if a cluster c has a mix of known and unknown samples, we calculate the uniformity of the cluster based on the known specimens. Following the ideas of RFoT, our cluster uniformity score is defined by the fraction of the most dominant class present in the cluster c:

$$U^c = \frac{|\max{(c^{known})}|}{|c^{known}|} \tag{5.2}$$

where U^c is the cluster uniformity score for the cluster c, $|c^{known}|$ is the number of known samples in the cluster, and the numerator is the number of samples that belongs to the most dominant known class in c. U^c specifies how uniform the given cluster c is based on the labeled data.

If the cluster uniformity score U^c is more than the threshold t, then we proceed to assign unknown specimens family labels in a semi-supervised fashion. That is, all the unknown samples are predicted to be the most dominant class in the cluster based on the known specimens ($\max{(c^{known})}$). If, however, the cluster uniformity score is less than the threshold t for a given cluster c, we form a new $\mathbf{X}' \in \mathbb{R}_+^{|c| \times m}$ that only contains the malware specimens present in that cluster (both known and unknown). Finally, $\mathbf{X}'$ is factorized again with NMFk. In the proceeding NMFk, k search range selected to be $[1, k^{opt}]$ with the step-size of 1. The above procedure is repeated until all the unknown samples are classified. In this setting, our leaf nodes in the hierarchical graph are the positions where at least one of the following exit conditions are met: no known specimens in the cluster (abstaining prediction), no unknown specimens are in the cluster (nothing to classify), or $U^c \geq t$ is true and we classify all samples in the cluster in a semi-supervised manner. The aforementioned procedure is summarized in Algorithm 3 and Figure 5.1.

In summary, looking at Figure 5.1 we can conclude that *HNMFk Classifier* is a wrapper to the NMFk algorithm, which exploits NMFk's ability to estimate the number of latent components, and performs factorization recursively to create a hierarchical graph where the semi-supervised classification is done at each leaf node. When our model finishes classification, any unknown samples that are left with the label -1 are said to be abstaining predictions, i.e. the model does not know their classes or rejects to make a prediction.

We also provide a toy example illustrating how *HNMFk Classifier* works in Figure 5.2. In this figure, we have a matrix $\mathbf{X} \in \mathbb{R}^{9 \times 3}$ (9 malware samples with 3 features). After factorizing $\mathbf{X}$ with NMFk, we get the latent factors $\mathbf{W} \in \mathbb{R}_{+}^{9\times k^{opt}}$ and $\mathbf{H} \in \mathbb{R}_{+}^{k^{opt}\times 3}$, with the estimated number of clusters $k^{opt} = 4$. Samples 5 and 6 are assigned to cluster 2. Sample 6, an unknown sample, is classified as family a. Cluster 3 contains only 2 unknown samples. Therefore, we classify samples 2 and 7 as abstaining. Cluster 1 contains the sample 1 (family a), 3 (family b), and 8 (unknown). Because this cluster has samples belonging to two different families (assuming that our cluster uniformity threshold is $t = 1$, i.e. threshold is met only when all the known samples in the cluster belongs to a single class), we create a new subset with these samples, such that $\mathbf{X}' \in \mathbb{R}_{+}^{3\times 3}$. We apply NMFk again on $\mathbf{X}'$, which estimates $k^{opt} = 2$, and sample 8 is classified as family b. When all samples are predicted, the computation is complete.

In our experiments, when testing HNMFk Classifier, we first use a small subset of the dataset (in a setup that does not reflect the real world) to understand the effects of using different hyper-parameters in our model, conduct ablation studies, and to observe the performance of our model with a decreasing quantity of labeled data. During our (more realistic) larger scale experiments, we use 2,898 classes of malware families (numbering more than 388,000 samples) with extreme class imbalance, and while including novel unknown malware samples during classification. Our method surpasses

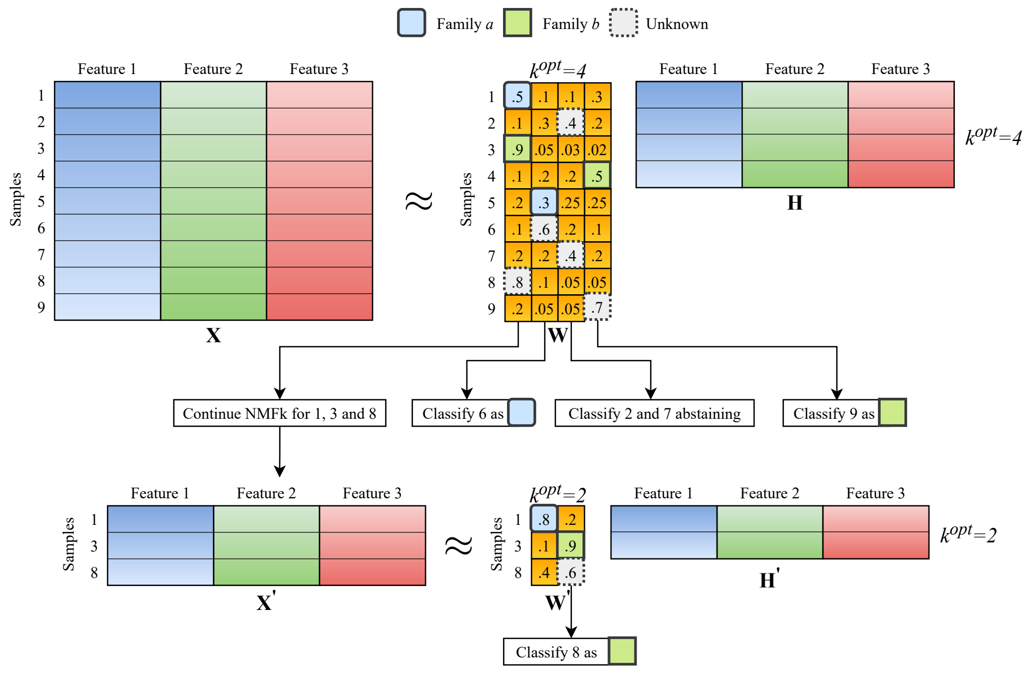


Figure 5.2: A toy demonstration of how HNMFk Classifier operates in a hierarchical fashion, and how the semi-supervised classification of the unknown malware specimens is performed via the clustering on the latent W matrices using the known samples.

the supervised baseline models *XGBoost* and *LightGBM* [41, 107]. We further extend these baselines with the *SelfTrain* algorithm to create strong semi-supervised models, which our approach still outperforms [179]. We also achieve better classification results compared to our Multi-Layer Perceptron (MLP) baseline [90]. To the best of our knowledge, we are the first to perform malware family classification over the EMBER-2018 corpus under realistic conditions such as the inclusion of the rare and novel families during our experiments, and our target number of family classes is around 29 times more than the previous work with the largest number of classes [94]

5.0.2 HNMFk Classifier: Performance Analysis

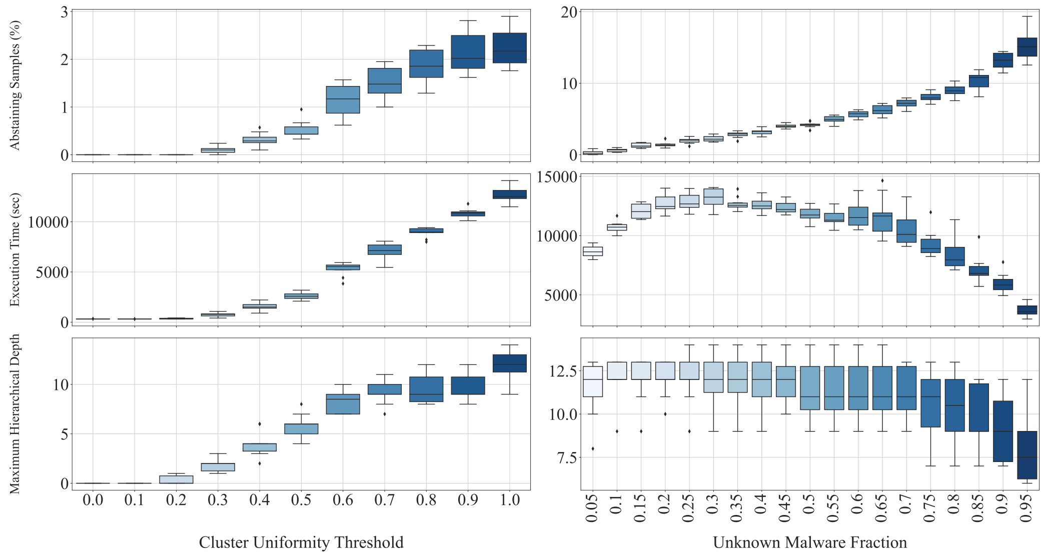


Figure 5.3: HNMFk Classifier's performance for abstaining prediction, execution time, and the maximum depth is shown as the cluster uniformity and the unknown malware fraction changes.

In this section we look at the performance of our method for different cluster uniformity thresholds, unknown malware fractions, and NMFk hyperparameter selections. Similar to prior work, we use a small subset of the dataset (an unrealistic data setup), during our analysis in this section. Each experiment is run 10 times on different random subsets of the dataset, to verify if the results are statistically significant using hypothesis testing. To this end, we report our results with a 95% confidence interval (CI) for each experiment.

Cluster Uniformity Threshold

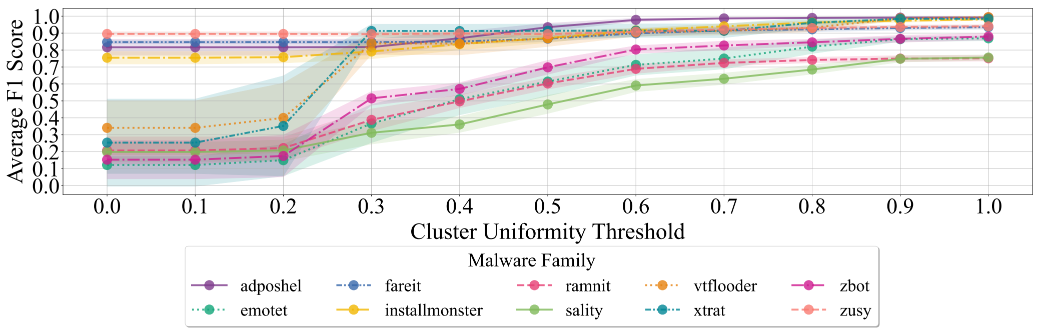


Figure 5.4: The performance of HNMFk Classifier is measured with the F1 score as the cluster uniformity threshold is changed. Each experiment is performed on 10 different random subset of the EMBER-2018 dataset, average is plotted with the 95% confidence interval.

We use a threshold value t, which measures how many labeled (known) specimens are needed to claim that all unknown specimens in this cluster are uniform, that is, from the same labeled malware family. This threshold allows us to determine whether to proceed further with clustering of the current data in the node with additional applications of NMFk. The left side of Figure 5.3 shows the percent of abstaining predictions, execution time, and the maximum graph depth (maximum number of edges between the root and a leaf node) as the cluster uniformity threshold t is changed. As t

increases, the percent of abstaining predictions rises, since the solution needs increasingly cleaner clusters. This reduces the number of specimens that we can classify with high certainty, and results in a higher number of abstaining predictions. The maximum graph depth also increases, alongside the higher execution time, since achieving cleaner clusters requires an increased number of separations. We show how the F1 score changes for each malware family in Figure 5.4. As the cluster threshold increases, the performance of the model improves for each malware family, and the results become more certain, as indicated by the narrowing confidence interval. Although the computation time increases, a higher threshold yields better inference results. Therefore, during the experiments we set the threshold to be $t = 1$.

Labeled Malware Data Scarcity Experiment

The process of labeling malware is expensive [148]; therefore, semi-supervised learning can help with obtaining good performance results when using a low quantity of labeled data. We investigate this by looking at how our model performs as the unknown malware fraction increases. Figure 5.5 displays the average F1 score for each malware family as the unknown malware fraction rises. Since our model can perform abstaining predictions, as the unknown

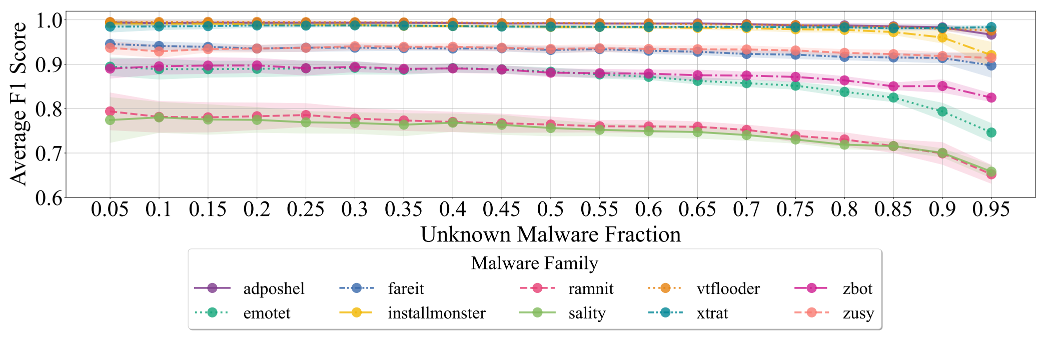
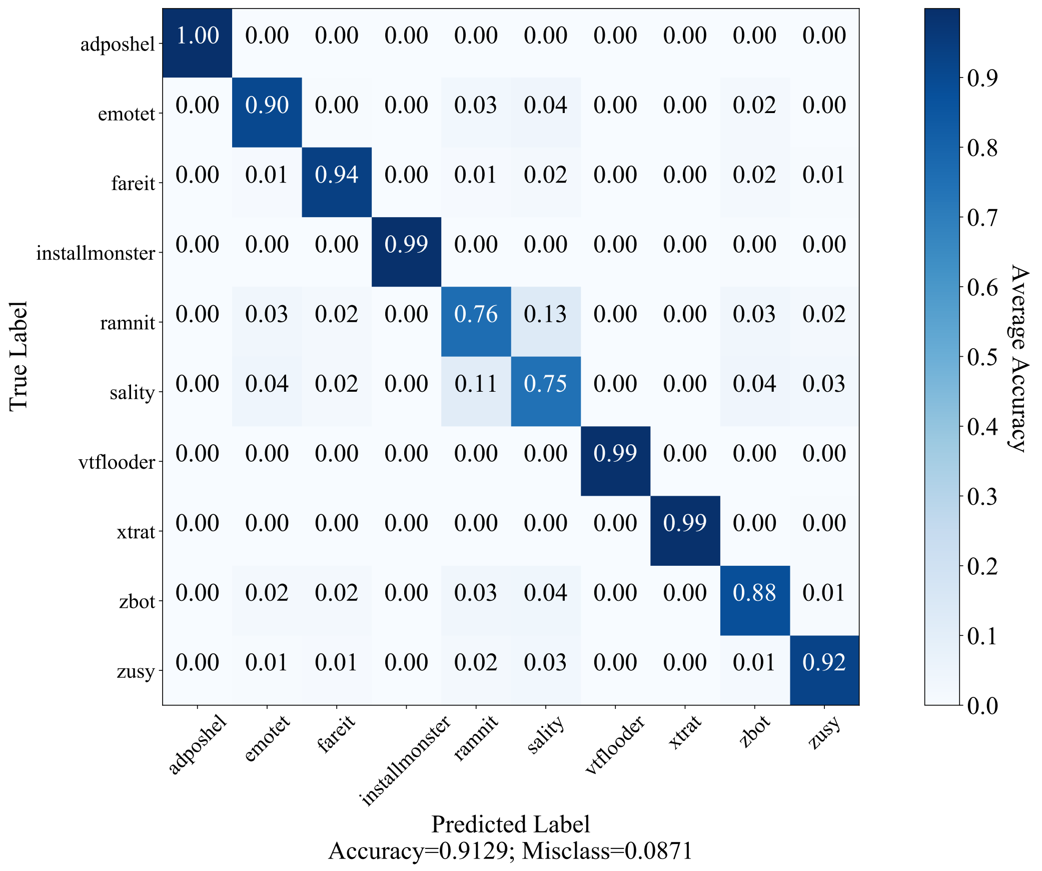


Figure 5.5: The performance of the HNMFk Classifer, measured with F1 score, remains relatively stable for each malware family as the unknown malware fraction increases (or the number of known samples decreases). Each experiment is run on 10 random subset of the dataset.

malware fraction increases, the performance of the model remains relatively stable. A lesser number of known malware samples means that our model to have a lesser number of references that can be used to classify the unknown samples. This results in higher number of abstaining predictions which in return helps with maintaining the performance (this can be seen at the right top of Figure 5.3). In Figure 5.5, we can also see that two malware families, Sality and Ramnit, yield lower F1 scores in comparison to the other families. Possible reasons for diminished performance on Sality and Ramnit include the fact that they are both *"file infectors"* (a category of malware which copies its code into other executables). It may be more difficult to classify this type of malware using the selected features, since some of the original PE metadata/file contents may not be changed when a file is infected.



	adposhel	emotet	fareit	installmonster	ramnit	sality	vtflooder	xtrat	zbot	zusy
adposhel	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
emotet	0.00	0.90	0.00	0.00	0.03	0.04	0.00	0.00	0.02	0.00
fareit	0.00	0.01	0.94	0.00	0.01	0.02	0.00	0.00	0.02	0.01
installmonster	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00	0.00
ramnit	0.00	0.03	0.02	0.00	0.76	0.13	0.00	0.00	0.03	0.02
sality	0.00	0.04	0.02	0.00	0.11	0.75	0.00	0.00	0.04	0.03
vtflooder	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00
xtrat	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00
zbot	0.00	0.02	0.02	0.00	0.03	0.04	0.00	0.00	0.88	0.01
zusy	0.00	0.01	0.01	0.00	0.02	0.03	0.00	0.00	0.01	0.92

Figure 5.6: Average F1 score when classifying 10 malware families is compared to other baseline models as the fraction of unknown malware increases. Each experiment is run on 10 random subset of the dataset.

We also show a confusion matrix in Figure 5.6, that showcases the accuracy of our method for each malware family. This figure displays the model accuracy for each malware family at a traditional know or training set size of 80% where the testing is performed at the remaining 20% of the samples. It can also be seen that the same two families, ramnit and sality, each return lower accuracy scores.

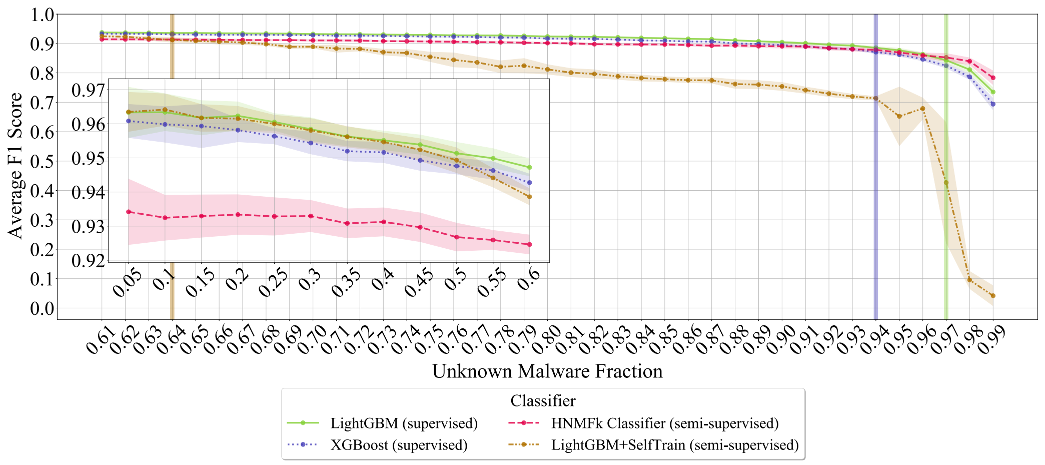
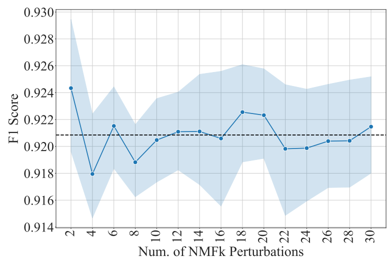


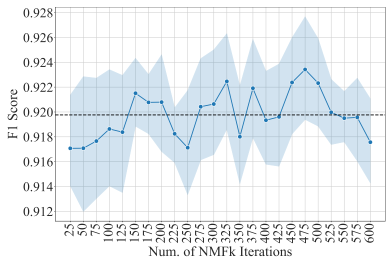
Figure 5.7: Average F1 score when classifying 10 malware families is compared to other baseline models as the fraction of unknown malware increases. Each experiment is run on 10 random subset of the dataset.

The average F1 scores obtained by the *HNMFk Classifier* with the changing unknown specimen fraction are also compared to the vanilla baseline models in Figure 5.7. Here, the unknown malware fraction point where the *HNMFk Classifier* begins to outperform a baseline model is shown with a vertical line. We use the supervised baseline models *XGBoost* and *LightGBM*, and a semi-supervised model *LightGBM+SelfTrain*. These traditional ML models do not have the ability to perform abstaining predictions. Therefore, they rely on an abundance of labeled data to perform well during testing. The *HNMFk Classifier* surpasses the average F1 score of *LightGBM+SelfTrain* at 0.64 unknown malware fraction. *XGBoost* is outperformed at unknown malware fraction 0.94, and *LightGBM* at 0.97. We also note that these models continue to perform relatively well as the known malware fraction drops because we are using a small and balanced subset of the dataset which contains the most populous malware families, making the problem easier. We will be further analyzing the performance of the baseline models and our approach with a realistic dataset setup. The experiments under real-world like setup will reveal that the performance difference between the baseline models and our method is even greater.

Hyper-parameter Analysis

In addition to the cluster uniformity threshold hyper-parameter of the *HNMFk Classifier*, we also provide our model with the hyper-parameters of NMFk. In Figures 5.8-a and 5.8-b we show that changes in the number of perturbations and NMF iterations do not have a large effect on the performance of our method. Figure 5.8-c displays the change in F1 score as the maximum k is increased for the k search of first NMFk. In this experiment, we choose the k step-size of 1, and begin searching at $k = 1$. The performance of the model continues to increase as the predicted k is approached. After the estimated k^{opt} is reached, the F1 score does not change, since we


(a)


(b)

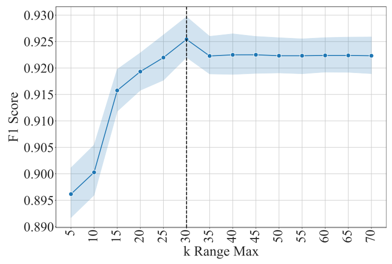

(c)

Figure 5.8: **(a)** Change in performance is measured using F1 score as the number of NMFk perturbations increased. It can be seen the effect to the overall performance as this hyper-parameter is changed is low, with average average F1 score of .92 and confidence interval .001. The difference between the highest and lowest point is .032. **(b)** Change in performance is measured using F1 score as the number of NMFk iterations increased. It can be seen the effect to the overall performance as this hyper-parameter is changed is low, with average average F1 score of .91 and confidence interval .0008. The difference between the highest and lowest point is .031. **(c)** Change in performance is measured using F1 score as the maximum k for of NMFk k search is increased. Here, k range is 1 through the maximum k value, with the step size of 1. It can be seen as the maximum k increases, the performance of the model improves. After the estimated k, increasing the value further does not change the performance.

will always choose the same k^{opt} in the first NMFk. These experiments indicate that we need to choose the initial k search range to be large enough to obtain a good initial clustering.

Realistic Malware Family Classification

Now that we have gained understanding into how our method performs with different hyper-parameters and settings, we will next use the more realistic data setup to show how our approach fares far better under real-world constraints. When ML-based malware defense and analysis solutions are used outside the research environment, they often encounter extreme class imbalance. At the same time, analysts do not have access to all possible malware samples, and threat actors continuously develop new pieces of malware. Therefore, ML-based systems are exposed to malware that has never been seen before. To this end, we analyze the performance of our method under a real-world like setting by exposing our model to prominent, rare, and novel malware families. In this section, we utilize all the malware families present in the EMBER-2018 dataset to conduct our experiment. The performance of *HNMFk Classifier* is compared to the supervised baseline models *LightGBM*, *XGBoost*, and *MLP*. We also create strong semi-supervised versions of *LightGBM* and *XGBoost* by wrapping them with *SelfTrain*. During the hyper-parameter tuning of *LightGBM* and *XGBoost*, we use the Python package *Optuna* to get the hyper-parameter suggestions for each trial [9], and for the construction of an optimal neural net-based classifier, *MLP*, we employed a *HyperBand Tuner* as an accelerated tuning algorithm [117]. The hyper-parameters of *LightGBM* was tuned using a stratified 20% subset of the training set over 65 trials and 3-fold stratified cross-validation. We used a stratified subset of the dataset because using the entire dataset for this model resulted in each trial taking approximately 2 days during tuning (it would have taken approximately 100 days to complete 50 trials for tuning). We used the objective *multiclass* with a 500 maximum number of iterations, and *gbdt* boosting type. The following hyper-parameters were tuned (ranges are shown in parenthesis): *min_data_in_leaf* (5-100 in log scale), *max_depth* (2-7), *bagging_freq* (0-5), *bagging_fraction* (.5-1.0), *learning_rate* (.001-.1 in log scale), and *feature_fraction* (.1-.7). For *LightGBM*, we have also tried the recommended hyper-parameters from the EMBER-2018 dataset [15], which did not yield better results when compared to our best trained model.

XGBoost was tuned using the entire dataset over 25 trials with stratified 3 fold cross-validation. We used maximum boosting rounds of 500 with the multi-class softmax objective function. The following hyper-parameters were tuned: *max_depth* (2-10), *eta* (.003-0.5 in log scale), *subsample* (.2-.7), *rounds* (10-300), *colsample_bytree* (.3-1.0), *colsample_bylevel* (.5-1.0), and *lambda* (.1-2.0).

The *HyperBand* framework has been widely used in the deep learning community for estimating the optimal parameters in a short amount of time. HyperBand is a variation of random search with explore-exploit theory to estimate best configurations within a given allocated time. The hyper-parameters utilized for model selection of the *MLP* were the number of depths of the neural network (1-10), number of nodes on each layer (1024-16000), optimization algorithm (SGD, Adam, RmsProp), and the learning rate (1e-4, 1e-1). We employed early stopping criteria on validation loss to avoid over-fitting.

Since our method's performance does not change dramatically with the change in hyper-parameters, we choose the hyper-parameters without tuning with 20 perturbations, 500 number of iterations, and k-range to be 1 through 100 with the step-size of 1 for the first iteration. We did verify, by inspecting the plot of the initial NMFk (similar to the Figure 2.5), that the estimated number of components was less than 100. If it had been close to 100, we would have re-started our experiment with a higher range. Finally, we chose the cluster uniformity threshold t to be 1, i.e. each cluster should have a single known class to be able to perform semi-supervised classification.

Table 5.1 compares our method to the baseline models. The *HNMFk Classifier*, a semi-supervised solution, outperforms all of the state-of-the-art models, which we used as baselines, with an F1 score of 0.80. Our approach outperforms the supervised methods, with the potential benefit of better generalizability and the need for less labeled data, due to the semi-supervised setting. We also surpass the strong semi-supervised version of *XGBoost* with *SelfTrain*. Notice that these baseline models were used to report benchmarks by prior studies. However, our experiment reveals the performance of these models under realistic conditions.

Model	**F1**	**Precision**	**Recall**	**Tune Time**	**Train&Predict Time**
HNMFk Classifier (semi-supervised)	**0.80**	**0.85**	**0.77**	5.77 days	7.91 days
HNMF2 Classifier (semi-supervised, ablation study)	0.77	0.82	0.74	5.77 days	2.83 days
XGBoost+SelfTrain (semi-supervised)	0.76	0.78	0.73	2.06 days	4.72 hours
XGBoost (supervised)	0.74	0.77	0.72	2.06 days	2.93 hours
LightGBM (supervised, tuned on stratified subset)	0.65	0.74	0.64	11.09 days	3.02 hours
MLP (supervised)	0.72	0.76	0.71	1.02 days	30 minutes
LightGBM+SelfTrain (semi-supervised)	0.64	0.69	0.61	11.09 days	9.44 hours

Table 5.1: HNMFk Classifier is compared against the state-of-the-art supervised classifiers. HNMFk Classifier, a semi-supervised method, surpasses the previous state-of-the-art models, which are supervised, in malware family classification. **Weighted** F1, Precision, and Recall scores are provided for multi-class classification with imbalanced data. The F1 scores of HNMFk Classifier and HNMF2 Classifier does not include the abstaining predictions (score includes the specimens where the prediction was not rejected)

Additionally, our method utilizes abstaining predictions (rejection to make a prediction), which other baseline models do not perform. We provide the metrics for the abstaining predictions in Table 5.2. The models that do not perform abstaining predictions always predict the novel specimens incorrectly since these samples belongs to a new class. The proposed ability to predict novel samples as *"other"* may still require the model to have seen the given specimen in the *"other"* class, which is not as effective as rejecting to make a prediction, which incorporates uncertainty in the model. In addition, as pointed out by Loi et al. [121], predicting specimens as *"other"* class often results in false predictions due to supervised models' common inability to learn patterns from a small number of samples. Our method's novel ability to reject making a prediction yields promising results in identification of novel malware. Interestingly, around 22% of the malware which we saw in the known set were also predicted as abstaining by the *HNMFk Classifier*. This 22% we referred as *false-abstaining*, since the specimens here belongs to classes that we had labels for. Importantly, around 42% of the novel malware (i.e. the malware which we did not see in the known set), are classified as abstaining. This 42% is referred as *true-abstaining* since our model did not have a reference label for these specimens in the known set. We also note that both true and false abstaining predictions would be caused by signatures or patterns extracted by NMFk being distinct from the labeled samples. Hence, it is possible that a detailed investigation and utilization of latent signatures can help to reveal characteristics that differ given specimen from the known samples (similar to the prior work in latent mutational cancer signatures [14]) and result in improvement of the abstaining predictions. This hypothesis motivated the development of MalwareDNA, which we will introduce as the next method in this dissertation.

In Table 5.2, for completeness, we also provide F1 scores for each baseline that is calculated only of the specimens that HNMFk Classifier did make a predictions (i.e. it did not abstain). Notice that the F1 scores of our baselines increase, even surpass our model in some cases, when the rejection to make predictions is not included in the score calculations. This result points out that the abstained samples are hard to correctly classify since our baselines yield lower scores when they are included (see the scored reported in Table 5.1). While the baselines falsely predicted the families for the harder specimens, HNMFk Classifier rejected to make a prediction and managed to maintain higher performance.

We also apply our ablation study, where the number of cluster selection

Model	Abstaining Seen (%)	Abstaining Novel (%)	F1 - (Non-reject)
HNMFk Classifier (semi-supervised)	22.06	**42.70**	0.80
HNMF2 Classifier (semi-supervised, ablation study)	**16.96**	34.16	0.77
XGBoost+SelfTrain (semi-supervised)	NA	NA	0.81
XGBoost (supervised)	NA	NA	0.80
LightGBM (supervised, tuned on stratified subset)	NA	NA	0.74
MLP (supervised)	NA	NA	0.79
LightGBM+SelfTrain (semi-supervised)	NA	NA	0.70

Table 5.2: HNMFk Classifier is compared against the state-of-the-art supervised classifiers. The ability of the HNMFk to discover novel families is shown. **F1 - (Non-reject)** column shows the F1 scores for the specimens that HNMFk Classifier did make a prediction on. Not applicable (NA) used at the cells where the case does not apply to the given model. **Abstaining Seen** refers to false-abstaining predictions, samples that belong to known classes that were seen in the training set. Differently, **Abstaining Novel** shows the true-abstaining predictions, where the specimen belongs to a class that were not seen before.

heuristic is turned off and rank-two factorization is used (i.e. $k = 2$ at each node). In table 5.1, we can see that the *HNMF2 Classifier* does perform better than our baseline models. However, the *HNMFk Classifier* outperforms this method, which points out that carefully choosing the number of clusters improves the separability and the overall performance during prediction. *HNMF2 Classifier* also reduces the percent of abstaining predictions, including the reduced percent of abstaining predictions on novel malware. We show additional results for ablation study on the automatic model selection below at Section 5.0.2.

Finally, note that in Table 5.1 we have also included the tuning time comparison between the *HNMFk Classifier* and the baseline models. The 5.77

days of tuning time listed for *HNMFk Classifier* comes from our performance analysis on selecting the cluster uniformity threshold t and understanding the effects of different hyper-parameter values of NMFk. We selected $t = 1$ for higher performance based on what we learned from the results of our experiments discussed in Section 5.0.2, and showed that the hyper-parameters of NMFk has a minimal affect on the model's performance. Note that the k selection procedure of *HNMFk Classifier*, which comes from the NMFk algorithm, is not a hyper-parameter adjustment, but a model selection, which is integrated in the algorithm [97]; therefore, it is not included in the tuning time. Instead, it is reported as the model training time. Our method takes about 8 days to complete running, which is significantly longer than our baseline models. In comparison to the traditional ML methods (in our case, the baseline models used in the experiments), our method is not a fast predictor. Instead, *HNMFk Classifier* is a bulk-classification method. The aforementioned 8 days computation time is the total inference time for the *HNMFk Classifier*. Therefore, our model is not suitable for real-time solutions, that is, for analysing of a single specimen at the time it comes in the system. Our method rather can be used for an accurate malware classification early in the labeling process. This is another motivation behind the development of

MalwareDNA, which is a real-time solution.

Ablation Studies

In our ablation studies we investigate the benefit of performing bulk classification and carefully choosing the number of clusters. To this end, during the first study we change the bulk classifier structure of our approach to form a more classical model, which we call the *HNMFk Classical Classifier*. During the second study, we ablate the automatic model selection heuristic from our method. The small subset of the EMBER-2018 dataset, is also used in our ablation studies in this section. As mentioned above, we use the top 10 malware families in the dataset with 1,000 specimens, each randomly sampled. Each experiment is run 10 times using a different random subset each time.

Bulk Classification:

To show that there is a benefit to doing bulk classification for our methodology, we compare the performance of the *HNMFk Classifier* to the *HNMFk Classical Classifier*, a model that does not perform bulk classification. This model also uses the known samples to form the hierarchical graph. We then predict the unknown samples separately over the hierarchical graph by following the edges, and computing similarity scores at nodes. For each of the n unknown malware samples, we obtain the cluster assignment by comparing the features $X_{i:}$ (ith sample) to the rows of the latent factor $\mathbf{H}$ using *cosine-similarity* score:

$$\text{cluster}(i) = \underset{0 \leq j \leq k^{opt}}{\arg\max} \left(1 - \text{cosine-distance}(\mathbf{H}_{ji}, \mathbf{X}_{i:})\right) \tag{5.3}$$

We follow each sub-clusters, comparing the features vector for the ith sample to $\mathbf{H}$ at each step, until we reach a leaf where we predict the label of the specimen i in a semi-supervised fashion. In Figure 5.9 we compare the F1 scores obtained from our ablation studies to *HNMFk Classifier* as the fraction of unknown samples change. From the figure, it can be seen that performing classification with *HNMFk Classical Classifier* yields unstable results, and our method *HNMFk Classifier* outperforms this model. This shows that bulk classification is beneficial in obtaining stable and accurate inference results.

Determination of the Number of Clusters:

The *HNMFk Classifier* utilizes the estimated number of components predicted by the NMFk algorithm to achieve good separability of malware fam-

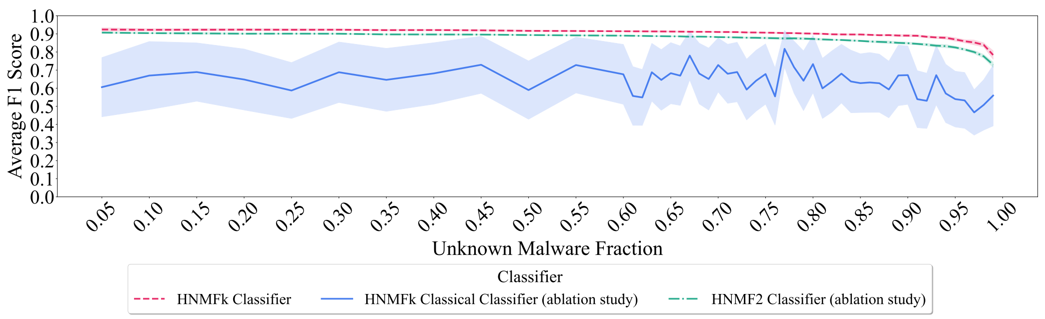


Figure 5.9: The performance of the HNMFk Classifier is compared to the other variants of our method from the ablations studies, as the fraction of the unknown malware is changed.

ilies. For the next ablation study, we look at the benefit of estimating k, or the number of clusters. During this study, we form another classifier named the *HNMF2 Classifier*, based on the previous work of Gillis et al. [85], which chooses $k = 2$ at each node, i.e. separate the data into two clusters at each step, until each known sample falls in separate leaf nodes.

In Figure 5.9, we also provide the results for the *HNMF2 Classifier*. Choosing $k = 2$ at each step performs almost as well as our approach. As also argued in [85], this result points out the benefit of hierarchical setting. Even if we make a bad separation of the samples due to rank-two factorization, the hierarchical approach will fix the separations in the proceeding splits. However, although slightly, our model outperforms the *HNMF2 Classifier*, which shows that choosing the number of components carefully using a heuristic is beneficial.

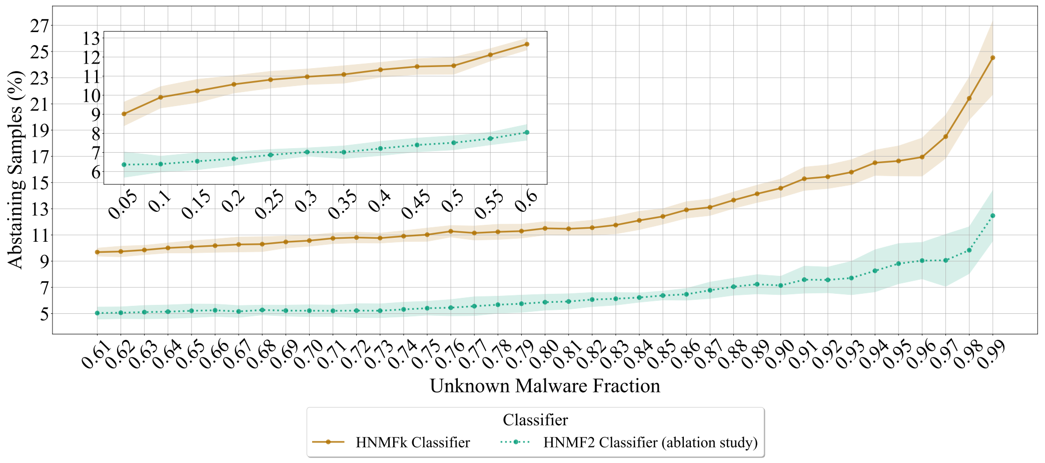


Figure 5.10: Percent of abstaining predictions for HNMFk Classifier and HNMF2 Classifier is compared as the unknown malware fraction in our dataset is increased.

Finally, Figures 5.10 and 5.11 illustrate the percentage of abstaining predictions and the total execution time, respectively, comparing the HNMFk Classifier and the HNMF2 Classifier as the fraction of unknown malware samples in our dataset increases. It is evident that for both models, the percentage of abstaining classifications increases with the rise in the fraction of unknown malware samples in our training or known set. Compared to the HNMF2 Classifier, the HNMFk Classifier yields a higher percentage of abstaining predictions, which results in more accurate decisions.

Similarly, Figure 5.11 shows that the HNMF2 Classifier operates faster than the HNMFk Classifier. This is expected since factorizing a matrix at each node when $k = 2$ is much quicker than searching for the optimal k value

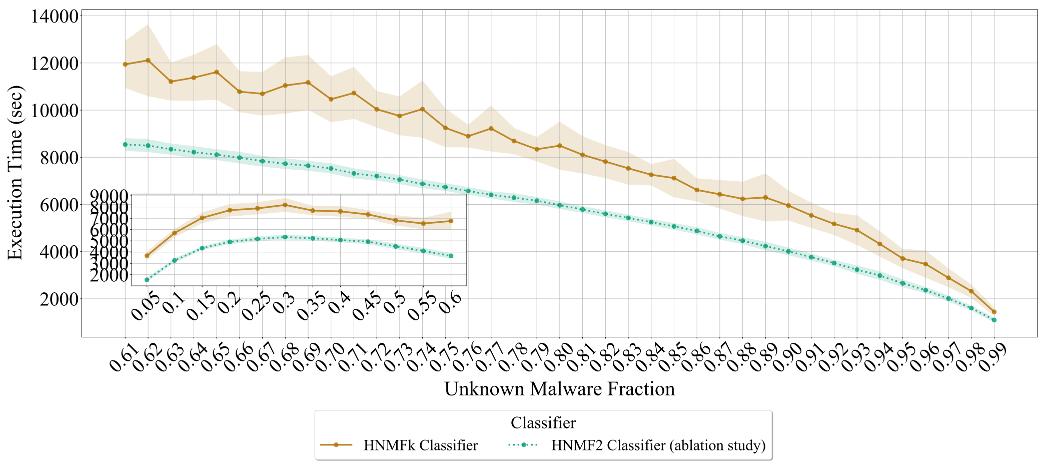


Figure 5.11: Execution time for HNMFk Classifier and HNMF2 Classifier is compared as the unknown malware fraction in our dataset is increased.

through automatic model determination in HNMFk Classifier. Additionally, we observe a decline in total execution time as the unknown malware fraction in the training or known set increases, which can be attributed to the reduced data size requiring analysis.

Algorithm 3: HNMFk Classifier($\mathbf{X}$, $\mathbf{y}$, k^{min}, k^{max}, r, t) - Semi-supervised Hierarchical Classifier

```
1:  known_samples= argwhere($\mathbf{y}$ != -1)
2:  unknown_samples = argwhere($\mathbf{y}$ == -1)
3:  $\mathbf{W}$, $\mathbf{H}$, $k^{opt}$ = NMFk($\mathbf{X}$, $k^{min}$, $k^{max}$, $r$)
4:  clusters = argmax($\mathbf{W}$, axis=1)
5:  for cluster in clusters do                              ▷ iterate over $k^{opt}$ clusters
6:      known_samples_c = intersect(known_samples, cluster)
7:      unknown_samples_c = intersect(unknown_samples, cluster)
8:      if len(known_samples_c) == 0 then ▷ no unknown samples to make
    prediction
9:          continue
10:     end if
11:     if len(unknown_samples_c) == 0 then             ▷ *abstaining* prediction
12:         continue
13:     end if
14:     class_counts = count(known_samples_c)
15:     cluster_uniformity = max(class_counts) / sum(class_counts)
16:     if cluster_uniformity ¡ t then
17:         $\mathbf{X}$_new = $\mathbf{X}$[cluster]          ▷ subset in $\mathbf{X}$, samples in the cluster
18:         $\mathbf{y}$_new = $\mathbf{y}$[cluster]          ▷ labels for the samples in the cluster
19:         $k^{max}$ = min($k^{opt}$+1, min($\mathbf{X}$.shape))
20:         y[cluster] = HNMFk_Classifier($\mathbf{X}$_new, $\mathbf{y}$_new, $k^{min}$, $k^{max}$, r, t)
21:     else
22:         classify_label = max(class_counts) ▷ dominant known class in the
    cluster
23:         y[unknown_samples_c] = classify_label
24:     end if
25: end for
26: return y
```

Chapter 6

MalwareDNA

MalwareDNA represents the pinnacle of our malware analysis capabilities, building upon the foundations set by both RFoT and the HNMFk Classifier. Developed from our experiences with both RFoT and HNMFk Classifier, MalwareDNA transitions from bulk classification to real-time classification capabilities. While the HNMFk Classifier demonstrated promising results in detecting novel malware, MalwareDNA achieves even greater accuracy and precision in identifying new malware families. Furthermore, while our experiments with the HNMFk Classifier were limited to malware families, with MalwareDNA, we have expanded the scope to include benign specimens. This approach not only aims at classifying malware families but also requires the model to distinguish benign samples from malicious ones simultaneously. This dual capability enhances the robustness and applicability of MalwareDNA in real-world scenarios.

Moreover, MalwareDNA incorporates new confidence metrics that significantly enhance its ability to detect novel malware and operate robustly under conditions of class imbalance. This model not only advances the technical sophistication of its predecessors but also refines their functionalities to perform more effectively in dynamic and challenging environments. Additionally, its capability for real-time classification makes MalwareDNA particularly well-suited for operational cyber-defense systems.

6.0.1 Building of Latent Signature Archive

An overview of how the signature archive is built is shown in Figure 6.1. MalwareDNA first applies NMF to the observational data $\mathbf{X}$ (Figure 6.1, **S1**). Recall that NMF is an unsupervised learning method based on a low-rank matrix decomposition [36]. For completeness and flow, we give a brief summary of NMF in this section as well. NMF approximately represents an observed non-negative matrix, $\mathbf{X} \in \mathbb{R}_{+}^{n \times m}$, as a product of two (unknown) non-negative matrices, $\mathbf{W} \in \mathbb{R}_{+}^{n \times k}$ whose k columns are the latent signatures

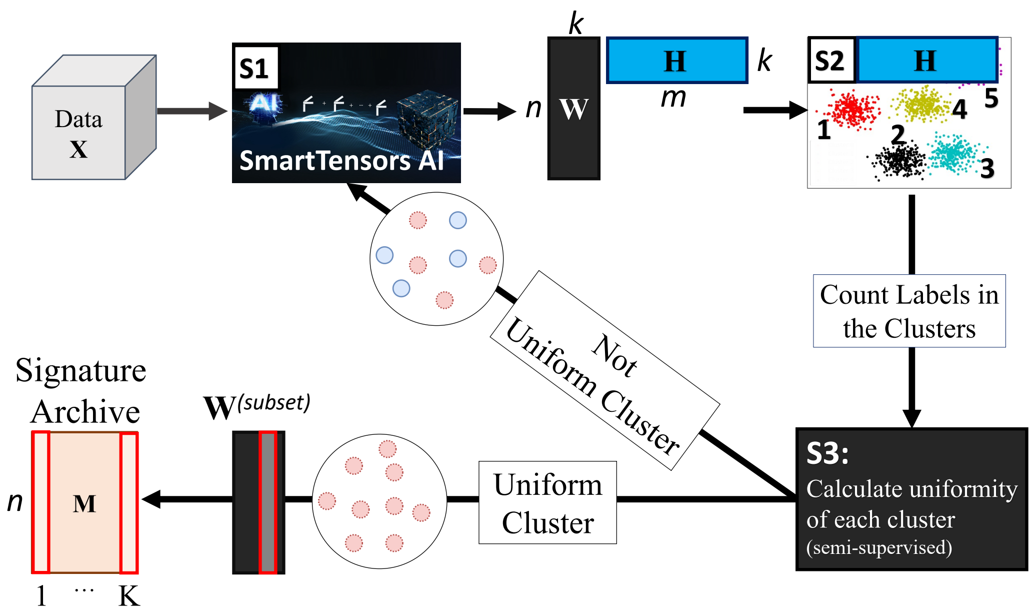


Figure 6.1: Overview of how the archive of latent signatures is built from multi-dimensional data in a hierarchical manner. The patterns from the data are first extracted (S1). These patterns have the corresponding clusters among the samples (e.g. malware specimens, S2). If we identify a cluster where each sample belongs to the same class (uniform), we place the patterns (or latent signatures) corresponding to this cluster into the archive (S3). Otherwise, we separate the mixed signatures of samples belonging to a non-uniform cluster by successive factorization (going back to S1).

each with n features, and $\mathbf{H} \in \mathbb{R}_{+}^{k\times m}$ whose rows are the activities of each one of the k signatures (latent features) in each m samples, where usually $k \ll m, n$. This approximation is performed via non-convex minimization with a given distance, $||...||_{dist}$, constrained by the non-negativity of $\mathbf{W}$ and $\mathbf{H}$: $\min||\mathbf{X}_{ij} - \sum_{s=1}^{k} \mathbf{W}_{is}\mathbf{H}_{sj}||_{dist}$.

The NMF minimization requires prior knowledge of the latent dimensionality k for accurate data modeling, which is usually unavailable [162]. Excessively small values of k lead to poor approximation of observables in $\mathbf{X}$ (*under-fitting*), while excessively large values of k fit the data's noise (*over-fitting*). In this work, we use *NMFk* that incorporates automatic model selection for estimating k [10, 171]. NMFk integrates NMF-minimization with custom clustering and Silhouette statistics, and combines the accuracy of the minimization and robustness/stability of the NMF solutions. A bootstrap procedure (i.e., generation of a random ensemble of perturbed matrices) is applied to estimate the number of latent features k. Recently, NMFk was applied to a large number of big synthetic datasets with a predetermined number of latent features, and it has demonstrated its superior performance of correctly determining k in comparison to other heuristics clustering techniques [34, 132]. MalwareDNA uses a publicly available implementation of NMFk [69][1].

After extracting the accurate factors $\mathbf{W}$ and $\mathbf{H}$, we apply a custom H-clustering method, the *Argmax* operator, to assign each sample (represented by the columns of $\mathbf{X}$) to one of the k signature clusters (**S2**) i.e. the label assignment for $X_{:,j}$ is given as $y_j = \text{argmax}(\mathbf{H}_{:,j})$ if $\max(\mathbf{H}_{:,j}) > \tau$ for some

[1]NMFk is available in `https://github.com/lanl/T-ELF`.

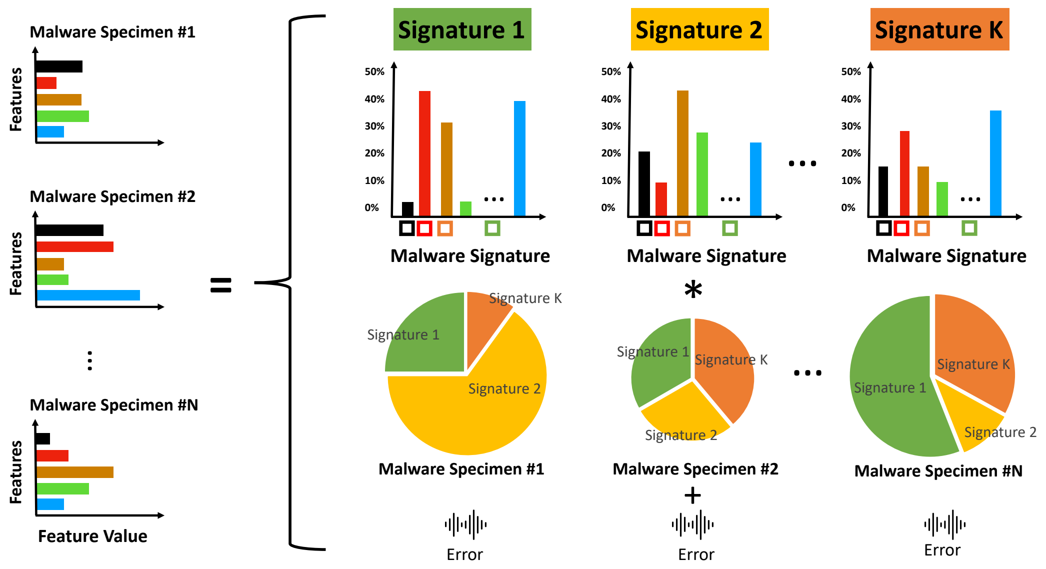


Figure 6.2: Illustration of a latent signature archive constructed by MalwareDNA: This archive comprises multiple signatures, each representing the characteristic patterns of a different malware family. When a new specimen is introduced, these signatures are utilized to characterize the sample by summarizing the extent to which its patterns match those present in the archive. This process helps in accurately identifying and classifying the malware based on its familial traits.

confidence probability/threshold τ. In each of these clusters, some of the samples may have different labels (non-uniformity) based on this confidence probability if $\max(\mathbf{H}_{:,j}) <= \tau$. We evaluate the uniformity of the samples in each cluster, determining whether all labels are the same (**S3**). Here, cluster uniformity score follows the same ideas of RFoT and HNMFk Classifier. Recall that the cluster uniformity score was formulated in Section 5.0.1.

When a uniform cluster is identified, we separate the samples of this cluster from the data, $\mathbf{X}$, and add the annotated (by the labels) cluster centroid, corresponding column of $\mathbf{W}$, to our archive of signatures $\mathbf{M} \in \mathbb{R}_{+}^{n \times K}$, where n is the number of features and K is the number of unique latent signatures. Otherwise, we continue with successive factorization in a hierarchical manner to separate the mixed latent signatures as shown in Figure 6.1. Further, Figure 6.2 depicts an illustration of an archive built by MalwareDNA.

6.0.2 Inference on the Latent Signature Archive via Projection

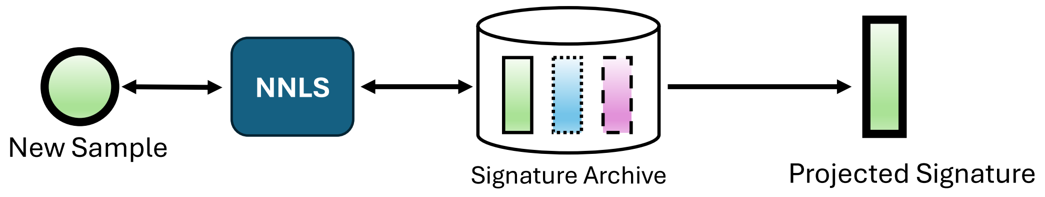


Figure 6.3: Illustration of how a new sample is projected onto the latent signature archive using Non-Negative Least Squares (NNLS) to obtain the projected signature.

We use the latent signature archive $\mathbf{M}$, after it is built, for inference - or classification - tasks. During testing for real-time inference, we project each new sample $\mathbf{x}$ onto the signature archive using Non-negative Least Squares Solver (NNLS)[35] where the optimization problem is given as $\arg\min_{h>=0} \|\mathbf{x} -$

$\mathbf{M}\mathbf{h}\|_2^2$ to extract coefficient vector $\tilde{\mathbf{h}}$. Figure 6.3 gives an illustration of this procedure. This allows us to perform real-time identification by representing each new sample as a combination of signatures recorded in the archive $\mathbf{x} \approx \sum_{i=1}^{K} \tilde{\mathbf{h}}_i * \mathbf{M}_i \implies \tilde{\mathbf{x}} = \mathbf{M}\tilde{\mathbf{h}}$ and estimating the accuracy, or similarity score, of this representation. We utilize the cosine similarity score of the NNLS projection of the new sample to the signatures $\mathbf{m} \in \mathbf{M}$ given as $S(\mathbf{m}, \tilde{\mathbf{x}}) = \frac{\mathbf{m}.\tilde{\mathbf{x}}}{\|\mathbf{m}\|_2 \|\tilde{\mathbf{x}}\|_2}$. Techapanurak et al. observed that cosine similarity is effective in identifying out-of-distribution samples [163], and Zhang et al. demonstrates cosine similarity as an effective metric to define the confidence of methods with reject-option capability [184]. We further define three different confidence metrics – Projection Similarity, Ensemble Voting, and Data Augmentation – using the cosine similarity scores from the NNLS projections.

6.0.3 Confidence Metrics

We have incorporated three different confidence metrics into MalwareDNA, each designed with unique discriminative capabilities. Here, "discriminative capability" refers to the model's effectiveness in accurately identifying whether a sample is truly novel when utilizing the reject-option. This involves determining if the sample identified as novel is indeed a new instance. Additionally, discriminative capability also encompasses the model's performance when it opts not to reject a prediction, focusing on the accuracy of the predictions made.

Projection Similarity

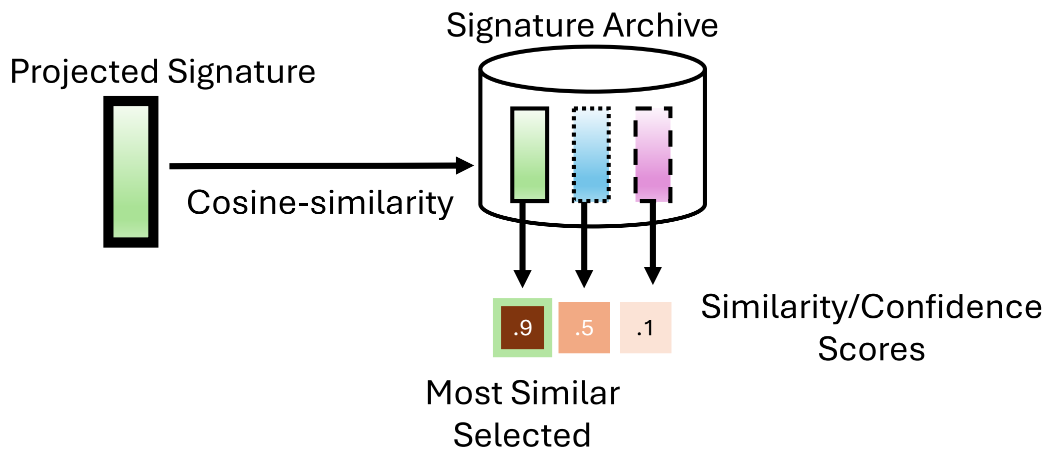


Figure 6.4: This illustration demonstrates how the projected signature is used to calculate cosine similarity scores against each signature in the latent signature archive. These signatures measure the model's confidence. The highest score that surpasses the specified threshold is selected for the prediction. If none of the similarity scores exceed this threshold, the sample is classified as novel, meaning the model would employ the reject option, responding with "I do not know."

We utilize the similarity scores, together with a threshold, τ, to define the malware family and novel malware family classification. Once we extract $\tilde{\mathbf{h}}$ based on NNLS approach discussed above, the prediction is then defined, using cosine similarity score S where $y_j = \underset{0\leq j\leq K}{\arg\max} S(\mathbf{M}_{:,j}, \tilde{\mathbf{h}})$. where the given prediction j is labeled $\mathbf{y}_j \in \{1, 2, \ldots, C\}$ for C classes. i.e. the most similar signature is selected based on distance measurement. When a signature possesses a similarity score above τ, the labels of the signature will be determined as the classification result. Differently, when the similarity score is below τ, the reject-option or abstaining classification will be selected where the returned label is -1. We provide an illustration of this process in Figure 6.4.

Ensemble Voting

Ensemble learning can further enhance the accuracy of our confidence calculation [184]. If we define a second threshold $\tilde{\tau}$ against the cosine similarity between $\tilde{\mathbf{h}}$ and each K signatures in $\mathbf{M}$, we can obtain votes for each class $\mathbf{y}_i^M \in \{1, 2, \ldots, C\}$. Given a sample $\mathbf{x}$, for each class C we can obtain V_C number of votes if the cosine similarity score between $\tilde{\mathbf{h}}$ and columns of $\mathbf{M}$ belonging to class C are above the given threshold $\tilde{\tau}$ ($\tilde{\tau} = 0.5$ in our experiments). We normalized the votes based on the number of signatures present for a given class in $\mathbf{y}^M$, such that $\hat{V_C} = V_C/|\mathbf{I}^C|$, where $|\mathbf{I}^C|$ is essentially is the number of latent signatures belonging to class C. This procedure for

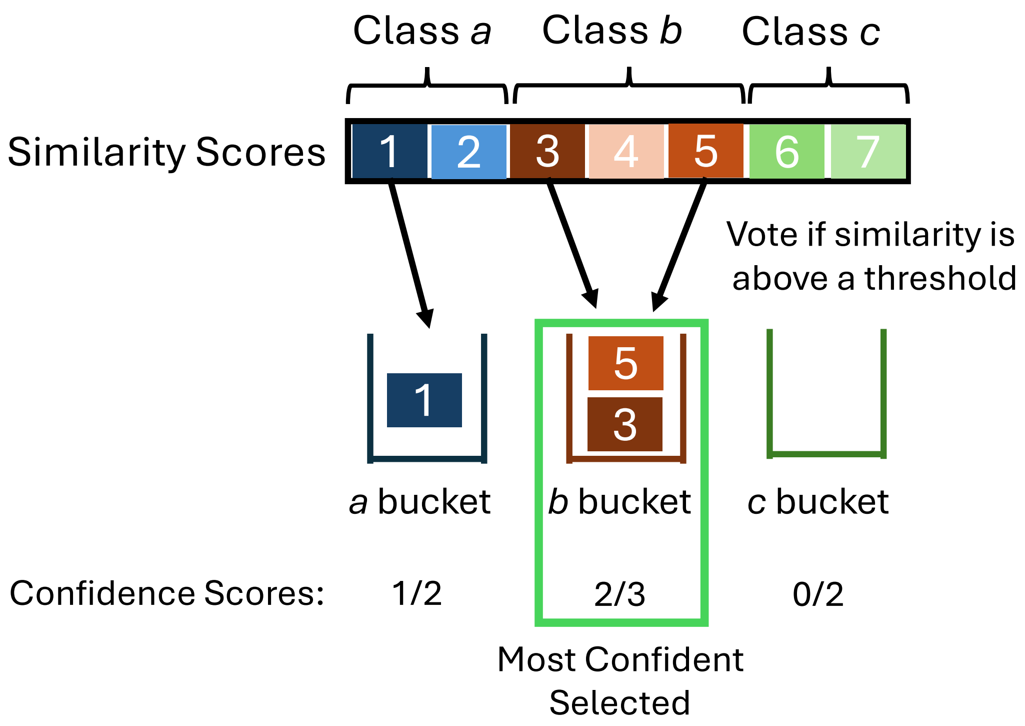


Figure 6.5: This illustration explains how ensemble voting is used to determine model confidence. Given a secondary threshold, $\tilde{\tau}$, the model assigns votes to buckets corresponding to each class if their cosine similarity scores exceed this threshold. The confidence score for each class is then calculated as the ratio of the number of votes to the number of samples in that class. The classification result is the class with the highest confidence score, provided that this score surpasses the primary threshold τ. If no confidence score meets this primary threshold, the reject option is employed.

defining confidence with ensemble voting is illustrated in Figure 6.5.

Data Augmentation

We also test our method with data augmentation to define the confidence, where the idea is confidence stability under perturbation for post-processing

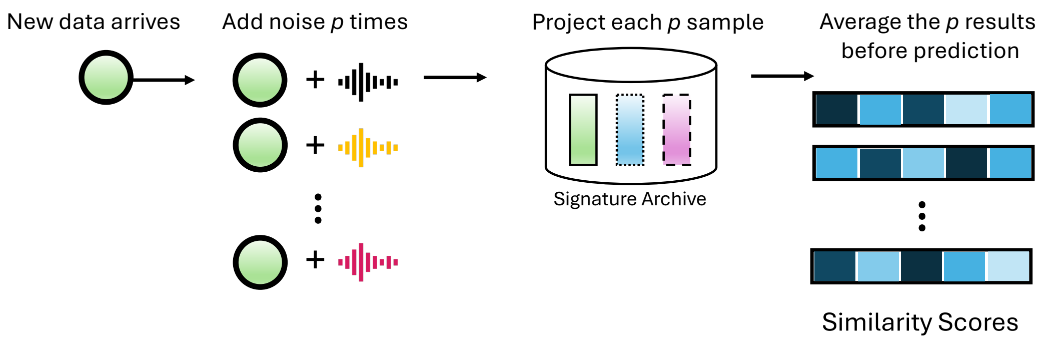


Figure 6.6: This illustration demonstrates how confidence is established using test-time data augmentation. When a new sample is received, MalwareDNA introduces noise through p different perturbations. Following the Projection Similarity confidence method described in Section 6.0.2, we calculate the cosine-similarity scores for each perturbation. The final vector of similarity scores is then obtained by averaging these p vectors of scores.

during testing time [23, 184], which is done with instance-level perturbations (*test-time data augmentation*) [24]. Here we add some error ϵ to $\mathbf{x}$ to generate p different perturbations $(\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, ..., \tilde{\mathbf{x}}_p)$ where $\tilde{\mathbf{x}}_i = \mathbf{x} + \epsilon|_{i=1}^{p}$, that is centered around $\mathbf{x}$ with distance $\|\epsilon\| = 0.015$ in our experiments), and average the corresponding cosine similarities of the predictions $S_{i=1}^{p}$ to define the confidence. The idea is that a truly confident outcome should remain stable under noise/perturbations, and that instance-level perturbations may yield more robust confidence measurements. Figure 6.6 gives an illustration of how the data augmentation at test level used to define the confidence. We apply this bootstrap approach 50 times in our experiments.

6.0.4 HPC, Multi-processing, and GPU Capabilities

MalwareDNA incorporates advanced computing techniques including multi-processing, high-performance computing (HPC), and graphics processing unit (GPU) capabilities. The integration of multi-processing and GPU functionalities is achieved through threading in Python, and with utilizing the CuPy library [136]. Specifically, for threading, we use Python's native `concurrent.futures` library. In parallel, HPC capabilities are facilitated by the implementation of OpenMPI, which is managed through the mpi4py library in Python [47, 48, 49, 50]. This section describes the algorithm designs employed to use multi-processing, HPC, and GPU resources.

MalwareDNA employs a hierarchical approach to applying NMFk, as discussed in Sections 2.0.5 and 2.0.6. As presented in Figure 6.1, MalwareDNA implements the NMFk operation recursively across hierarchical levels when building the signature archive, where each application of NMFk targets a specific subset of the data. This methodology generates a graph structure in which each node represents an NMFk operation performed on a data subset. Example visualizations of such graphs, using real malware data, were demonstrated in Figures 2.6 and 2.7. Within this graph, nodes at the same graph depth operate independently, as each is responsible for processing a

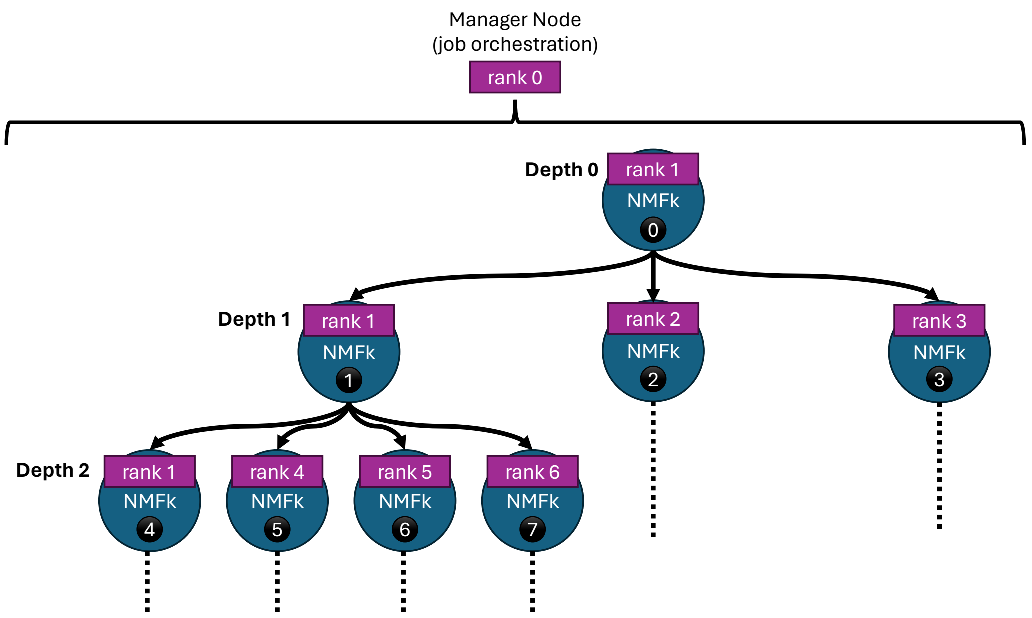


Figure 6.7: MalwareDNA constructs a hierarchical graph in which each node executes NMFk operation. The computation of these NMFk operations is distributed across multiple worker MPI ranks, which correspond to compute nodes in a HPC system. This distribution is orchestrated by a manager rank, which is responsible for job coordination and efficient resource allocation.

distinct subset of samples derived from its parent node. These subsets are obtained by the clusters found by NMFk. Figure 6.7 provides an additional visualization that illustrates how these computations are distributed across multiple HPC ranks, or compute nodes within an HPC system. Note that here we use the term distributed computation to describe how we make use of multiple compute nodes in an HPC system to run embarrassingly parallel tasks. This means, this implementation is not distributed in terms of data, where chunks of the data is processed across different HPC compute nodes.

In Figure 6.7, it is important to note that Rank 0 serves as the manager node, responsible for job orchestration. Here the term Rank refers to a compute node in an HPC system. With OpenMPI, a communicator obtained and it allows communication among different ranks of a HPC system. The initial NMFk computation, conducted on the entire dataset, is carried out by Rank 1 (Node 0) at Depth 0. This initial operation, as illustrated, separates the malware specimens in our data into three separate clusters. Subsequently, at Depth 1, Ranks 1, 2, and 3 begin their respective NMFk operations on each cluster simultaneously. For illustrative purposes, the operation at Node 1, conducted by Rank 1, completes ahead of Nodes 2 and 3, identifying four distinct clusters. While Nodes 2 and 3 continue processing, Ranks 1, 4, 5, and 6 simultaneously initiate NMFk operations on these newly identified clusters. By the time Depth 2 is reached in this demonstration, a total of six compute nodes are actively engaged in performing NMFk operations concurrently. The manager node here, which is at Rank 0, is responsible of managing the job queue.

Figure 6.8 illustrates the job management process for distributing the NMFk computation across multiple HPC nodes. In the first section of the figure, the manager node is shown dispatching jobs to four worker nodes.

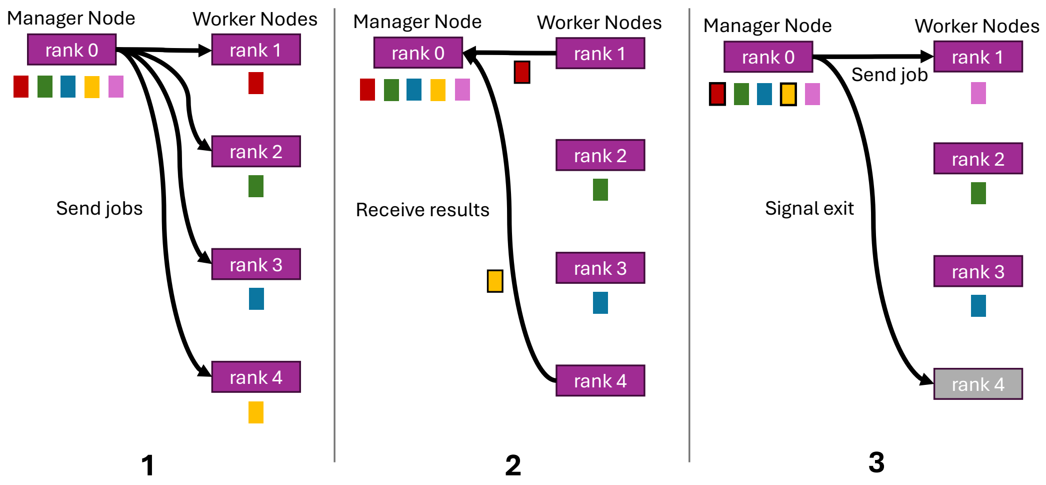


Figure 6.8: The illustration of job management for distributed MalwareDNA is shown in this figure. Rank 0, serving as the manager node, is responsible for orchestrating the job queue. This includes dispatching jobs to worker nodes, receiving completed jobs from them, processing the results, and ultimately, if no jobs remain, terminating the worker nodes.

Jobs are sent to specific worker nodes using the `comm.isend` command and are received by the worker nodes from the manager using `comm.irecv` in mpi4py, which initiates the communication request. Here, `comm` refers to an MPI communicator. The compute nodes wait for this communication request to complete before proceeding. Communication utilizes a buffer to handle the transfer of Python dictionary objects across multiple HPC nodes. The manager node periodically checks the status of the worker nodes (once every second, unless busy with another communication). As shown in the second part of the figure, Ranks 1 and 4 complete their tasks and notify the manager node of their status changes, prompting the manager to collect the results using `comm.irecv`. At this point, the completed NMFk operations may result in the identification of new clusters for hierarchical factorization, potentially adding more jobs to the queue. However, for the purposes of this illustration, it is assumed that no new jobs are added. In the final section of the figure, the manager node sends the last job to Rank 1 and issues an exit signal to Worker Rank 4. Rank 4 receives the exit signal through `comm.iprobe` and subsequently terminates. `comm.iprobe` is a non-blocking communication, that is, if the signals status does not exist, the node does not wait and continue its computation with remaining of the logic. In this situation, the logic will result in the worker node to come back to exit check portion of the system, and eventually receive the exit signal.

Note that in this setup, at Node 0, Rank 0 is tasked with performing NMFk on the entire dataset within a single compute node. While each subgraph processes a distinct subset of the data, thereby reducing the volume analyzed at each subsequent depth, the initial computation at Depth 0 involves handling a large dataset. Consequently, additional HPC capabilities have been integrated to distribute the NMFk operation across multiple compute nodes. NMFk conducts factorization for various K values, where K

denotes the matrix or tensor rank (distinct from MPI rank). This rank represents the number of linearly independent columns and rows, or the optimal number of latent patterns (clusters). During the initial application of NMFk on significantly larger datasets, it is advantageous to distribute the computation across multiple HPC nodes to facilitate the exploration of K values. As demonstrated in Figure 2.5, we have previously illustrated how NMFk evaluates several K values, determining the optimal one by employing heuristics for stability and accuracy in a bootstrap approach.

Figure 6.9 illustrates the execution of the distributed NMFk operation. In this version of the HPC algorithm for NMFk, there is no designated manager node; instead, each node independently performs parts of the NMFk operation. At Rank 0, or compute node 0, the process begins with a K search space defined as `Ks=[1,2,3,4,5,6,7,8,9]`. This configuration mandates that factorization be executed for each matrix/tensor rank listed, to heuristically identify the optimal number of clusters that yield the best sample separation. The system broadcasts this search space to each compute node. Note that the computation times for lower ranks are typically shorter than those for higher ranks. To enhance efficiency and resource utilization, matrix/tensor ranks are distributed across nodes in a manner that ensures each

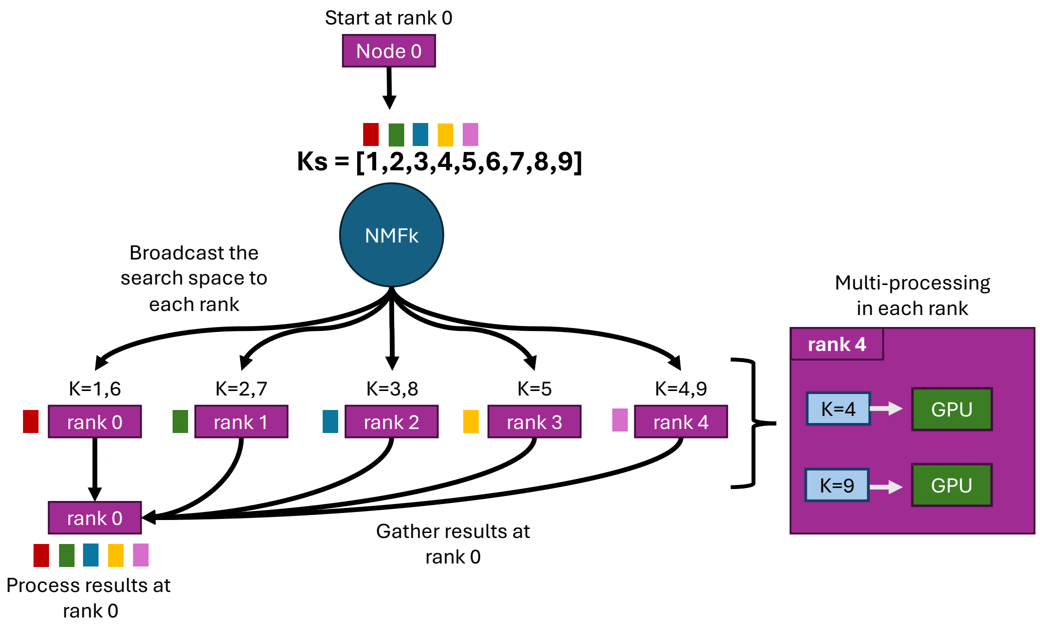


Figure 6.9: The figure illustrates the distribution of each K value, representing the matrix or tensor rank, across multiple HPC nodes. Additionally, it demonstrates how multi-processing within each node enables the concurrent factorization of different matrix/tensor ranks on each GPU. This method not only leverages the computational power of multi-node environments but also optimizes the utilization of GPU resources to expedite the factorization process.

node processes both lower and higher ranks. This strategy prevents scenarios such as Rank 0, receiving `Ks=[1,2,3]`, completing its tasks significantly earlier than Rank 1, which might receive `Ks=[4,5]`, thereby causing Rank 0 to idle while waiting for Rank 1 to finish. In our depicted example, Rank 0 is assigned `Ks=[1,6]`. Once all compute nodes complete their respective factorization processes, Rank 0 collates the results and analyzes them to determine the ideal tensor/matrix rank.

Finally, note that in Figure 6.9 that MPI Rank 4 was assigned two K values, `Ks=[4,9]`, for factorization. On each compute node, we employ multi-processing techniques to concurrently factorize these K values. This concurrency can be achieved through multiple CPU processes, or by utilizing threading to engage multiple GPUs within a system. In Figure 6.9, we provide an illustration of how Rank 4 concurrently utilizes two GPUs to factorize `Ks=[4,9]`. In summary, our algorithm has been specifically designed to leverage HPC systems, multi-processing, and GPU capabilities to accelerate linear algebra operations. This approach effectively streamlines the processes required to construct an archive of latent signatures with MalwareDNA.

6.0.5 MalwareDNA: Performance Analysis

In this section, we showcase the performance of MalwareDNA across different experimental setups designed to examine the distinct characteristics of the method. We begin with our initial analysis outcomes and progressively build the narrative towards more specialized dataset setups for a thorough examination of the method. While we will point out each dataset settings as we showcase the results as a short reminder, recall that the summary of experimental setups can also be found in Section 2.0.8.

All experiments involving MalwareDNA, including the ones involving baselines, were conducted on the LANL's High HPC system named Chicoma. Each compute node within Chicoma is equipped with four NVIDIA A100 GPUs, 252 GB of RAM, and an AMD EPYC 7713 64-Core processor. To construct the signature archives using MalwareDNA, each experiment and its corresponding cross-validation process utilized four nodes of Chicoma. For example, for an experiment with MalwareDNA where we run ten cross-validations (CV), where data was randomly sampled on each CV, we used total of forty nodes ($\#$ nodes $= 4$ nodes$*10$ CV) to complete the experiment. This setup allowed for the distribution of NMFk operations in MalwareDNA across multiple nodes. Additionally, multi-processing was employed to use all four GPUs on each node, thereby parallelizing the search for the optimal rank within each node with NMFk. In contrast, the prediction tasks using the MalwareDNA archive were executed on a single Chicoma node utilizing a single process. The baseline models were also tuned and trained on a single node of Chicoma exploiting the GPUs and multi-processing capabilities.

The First Look at the Performance

In this section, we begin by showcasing the capabilities of MalwareDNA. The purpose here is to provide a background that will support our investigation into the results in subsequent sections, which will delve deeper into the analysis methods.

To illustrate the capabilities of MalwareDNA, we first randomly sample 1k benign-software and malware specimens from four families (ramnit, adposhel, emotet, and zusy) using a popular benchmark dataset, EMBER-2018 [15]. We select ramnit to represent a malware novel/unseen family. The results shown in this section used MalwareDNA's Projection Similarity method to define the confidence.

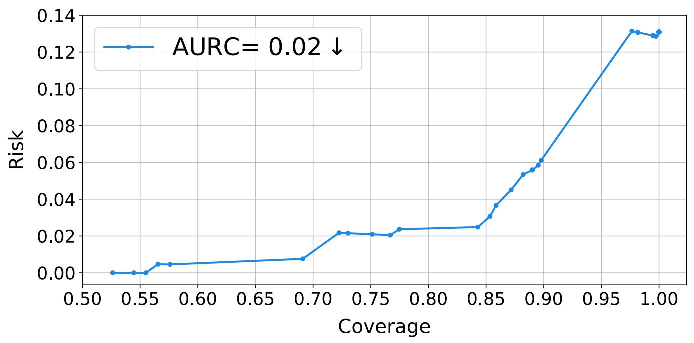


Figure 6.10: Risk-Coverage (RC) curve when classifying malware families and the benign-software, together with the area under the RC (AURC).

The performance of our method is first reported with AURC [53] in Figure

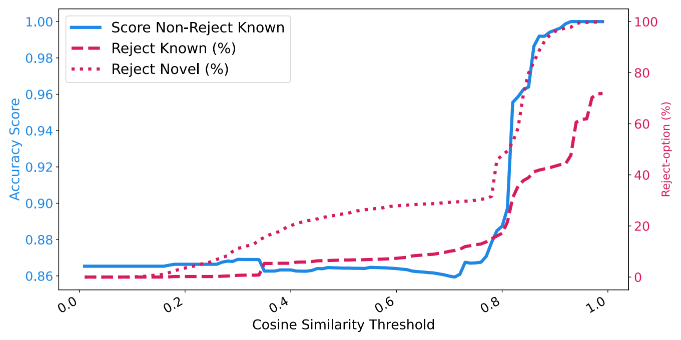
Model	F1	Precision	Recall	Rejection Seen	Rejection Novel
MalwareDNA (ours)	**0.975**	**0.975**	**0.977**	15.70 %	**100.00 %**
XGBoost	0.416	0.699	0.510	NA	NA
LightGBM	0.297	0.749	0.338	NA	NA
XGBoost+SelfTrain	0.096	0.258	0.108	4.34 %	18.09 %
LightGBM+SelfTrain	0.096	0.078	0.197	**2.89 %**	17.14 %

Table 6.1: Performance of MalwareDNA compared to baselines. Rejection Seen provides the false rejection predictions for the samples that belongs to known classes. Rejection Novel is the true rejection predictions for the samples that belongs to a novel malware family. XGBoost+SelfTrain and LightGBM+SelfTrain achieve AURC score of 0.654 and 0.651.

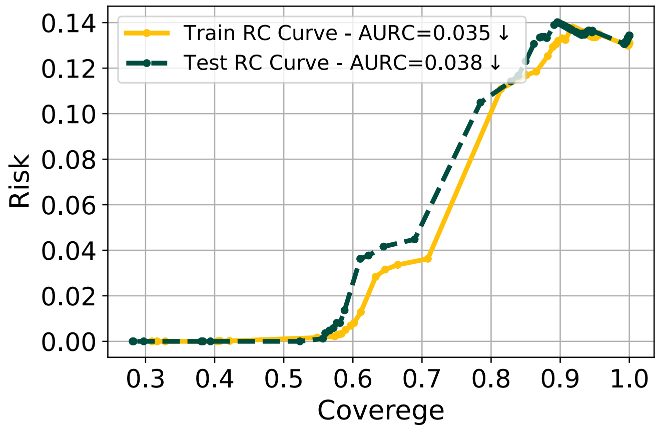
6.10. Recall that AURC models the trade-off between the coverage (the number of samples for which the non-rejecting predictions were made) and the risk which is measured with 0/1-loss. AURC score is reported between 0 and 1, and lower AURC is preferred over higher AURC. MalwareDNA achieve AURC of 0.02 when classifying the three malware families and benign samples. Our score indicates that we can achieve high coverage with minimal increase in the risk (false rejection predictions).

At 84.3% coverage, MalwareDNA achieves an F1 score of 0.975 when classifying the malware families and the benign-software and 100% true-rejection predictions for the chosen unseen family ramnit, which illustrates our method's ability to identify novel malware families (Table 6.1). In Table 1, we also baseline our method against the state-of-the-art supervised malware classifiers XGBoost [42] and LightGBM [108]. We further extend these baselines with the SelfTrain [179] algorithm to create semi-supervised models. We note that the previous work has used these models to report benchmarking against this dataset [15, 124]; however, we expose these models to a more challenging task of classifying malware families, separating them from benign samples, and detecting novel malware families all at the same time. Our baselines are tuned using Optuna [9] over 100 trials with 5-fold stratified shuffle cross-validation. Our benchmarking against the baseline models and the poor performance of these models, points out both the difficulty of the task, and MalwareDNA' unique capability to both accurately detect malware, classify families, while simultaneously detect novel malware families.

We have expanded these results to include 20,000 samples, randomly selected from the EMBER-2018 dataset with no benign specimens included. As illustrated in Figure 6.11-b, we achieved a notable AURC score of 0.038. This indicates that at approximately 50% coverage, the accuracy rate is around 0.99, and 99% of Ramnit samples are correctly identified as novel, as shown in Figure 6.11-a. During these experiments, we processed a data stream, classifying an average of 18 specimens per second using a single processing core, which demonstrates the real-time classification capabilities of MalwareDNA.



(a)



(b)

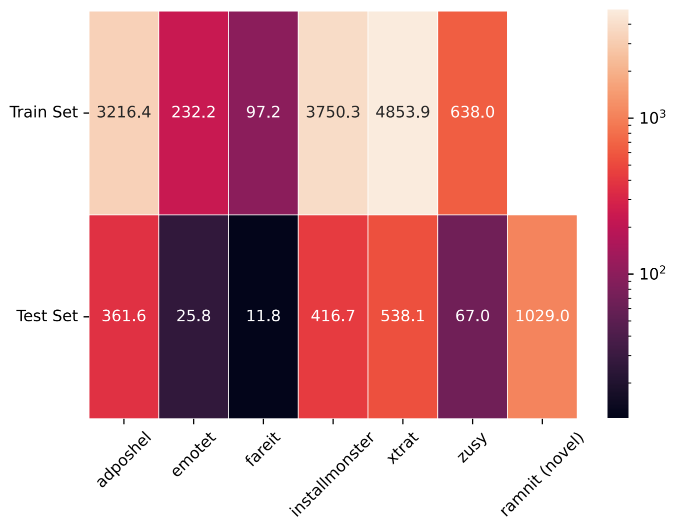
Figure 6.11: **(a)** Accuracy score, and percent of abstaining predictions as the cosine-similarity threshold is raised is shown for Reject Known (false rejection), Reject Novel (true rejection), and non-rejected known specimens. **(b)** Risk-Coverage (RC) curve when classifying malware families and the benign-software, together with the area under the RC (AURC).

Additionally, as depicted in Figure 6.11-a, when the cosine similarity threshold approaches 1, our method accurately identifies almost all novel samples. However, nearly 80% of known samples are incorrectly identified as novel. Thus, there is a clear trade-off between the desired coverage and the goal of identifying novel specimens.

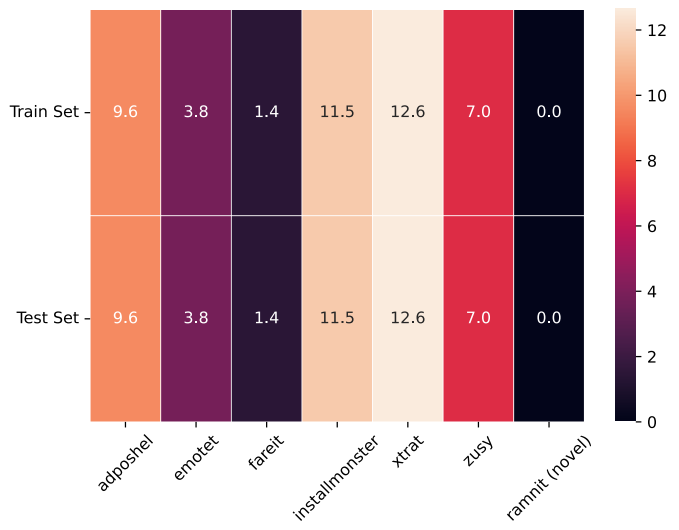
While these initial results highlight the promising capabilities of the method, we will next explore the distinct characteristics of MalwareDNA. Additionally, it is important to note that no confidence intervals are reported in this section. We will include cross-validation results in all subsequent sections to illustrate the statistical significance of the findings, and report our results with confidence intervals.

Confidence Methods

In this section, we investigate the performance of our various confidence methods: Projection Similarity, Ensemble Voting, and Data Augmentation, which are utilized to define the reject-option. The results presented here follow the dataset setup for addressing the class imbalance problem as described in Section 2.0.8. We ten times randomly sampled from seven malware families, selecting three to represent the rare families, which are randomly



(a)



(b)

Figure 6.12: Dataset distribution for the experiments for ”Class Imbalance” and ”Confidence Methods” is shown. **(a)** The heatmap displays the average number of samples in the train and test sets for each malware family. **(b)** For each of the 10 experiments, we randomly sample the dataset. This heatmap illustrates the confidence intervals for the samples in the train/test split for each malware family.

down-sampled. Notably, Ramnit is chosen to represent the novel family. Figure 6.12 illustrates the distribution of training and testing data.

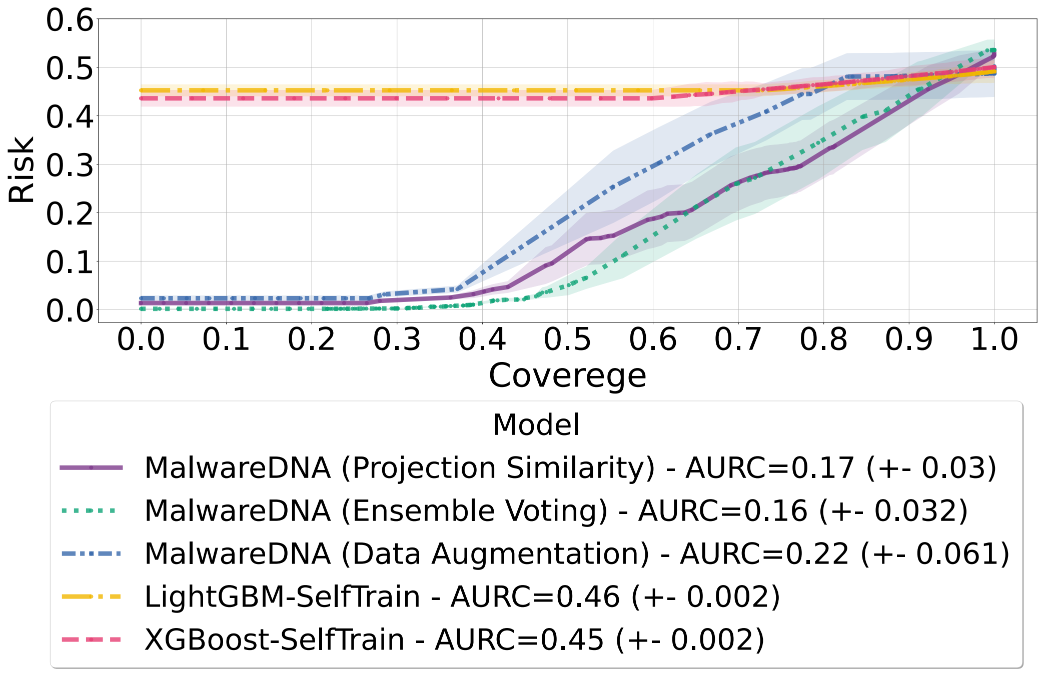


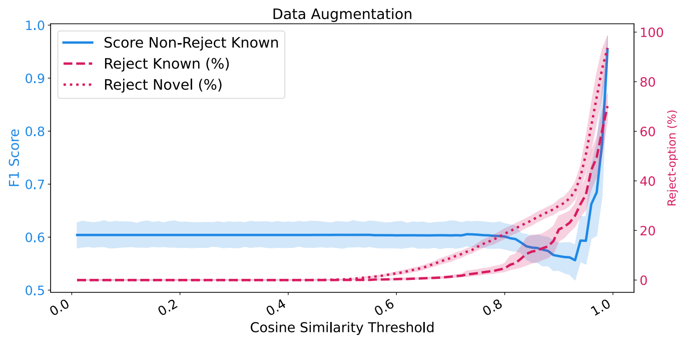
Figure 6.13: Risk-Coverage (RC) curve when classifying malware families and novel malware together with the area under the RC (AURC) for different MalwareDNA confidence metrics and our semi-supervised baselines.

We compare various confidence methods using AURC scores, as shown in Figure 6.13. Here, risk is quantified as $1 - F1$ score. Notably, this comparison includes two semi-supervised baselines. These semi-supervised models are not designed for reject-option. However, for comparison to our semi-supervised method, we use the prediction probabilities of the semi-supervised baseline methods with the same reject-option threshold to define the abstaining predictions. That is, when the prediction probability is less than the given threshold, given specimen is abstained from the prediction.

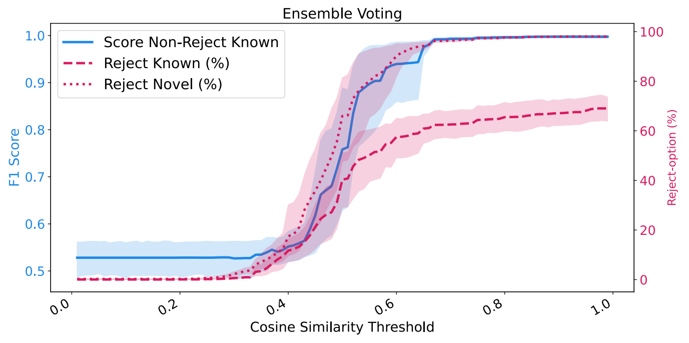
MalwareDNA with Ensemble Voting achieves the best AURC score of 0.16 when classifying malware families. It is also observed that Ensemble Voting maintains a lower risk (higher F1 score) as the coverage rate increases (i.e., the fraction of abstaining predictions decreases) up to a coverage rate of approximately 0.65. Additionally, across most coverage rates, MalwareDNA with each confidence metric consistently yields a lower risk score compared to our semi-supervised baselines.

Since our baselines are not designed with a reject-option, their risk levels remain unchanged for about half of the coverage rates. However, their coverage begins to decline after the coverage rate surpasses 0.6, as they start to generate incorrect predictions.

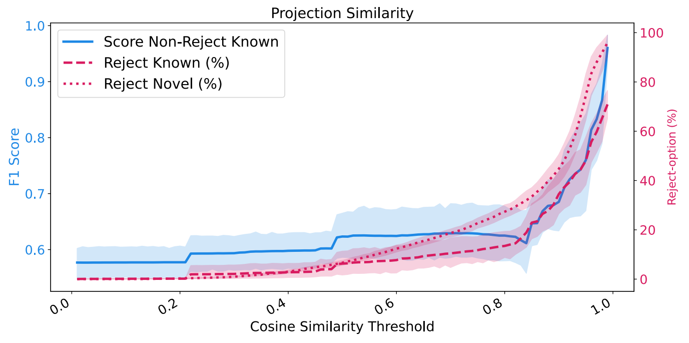
We next compare the confidence metrics using F1 scores and the percentage of abstaining predictions for known and novel samples across different confidence thresholds, as illustrated in Figure 6.14. In Figure 6.14-a we observe that with Data Augmentation, the model's performance remains stable until reaching a high confidence threshold, at which point there is a sharp increase in both the score and the percentage of abstaining predictions. It



(a)



(b)



(c)

Figure 6.14: F1 scores and percent of abstaining predictions (reject-option) for different cosine similarity score thresholds are shown. We include score for known samples, and the reject-option for known and novel samples. **(a)** Data Augmentation **(b)** Ensemble Voting **(c)** Projection Similarity

is also important to note that there is minimal difference in the percentage of abstaining predictions between novel and known samples, suggesting that Data Augmentation may not be the ideal confidence metric for novelty detection.

Figure 6.14-b illustrates the results for Ensemble Voting. This confidence metric significantly outperforms others in distinguishing between novel and known samples at much lower confidence thresholds, demonstrating its robustness in classification accuracy and distinguishing between novel and known samples. At a confidence threshold of about 0.6, Ensemble Voting misclassifies approximately 60% of the known samples as novel, indicating a conservative approach towards classification that prioritizes minimizing false negatives. While this high rate of false positives for known samples might seem detrimental, it reflects the method's stringent criteria for classifying samples as known, thereby enhancing its ability to detect novel threats effectively.

Conversely, Ensemble Voting correctly identifies almost all novel samples as novel, achieving an F1 score of around 0.9. This high F1 score underlines the method's ability in recognizing new malware variants, which is critical for maintaining system security against the most recent or novel threats.

The ability to accurately identify novel samples while maintaining a high F1 score is indicative of the sophisticated underlying model architecture that effectively balances sensitivity and specificity. This performance is particularly noteworthy because it demonstrates the potential of Ensemble Voting to act as a reliable safeguard in operational settings where the cost of missing a novel threat can be substantial. By setting the confidence threshold at 0.6, the system ensures a rigorous vetting process, substantially reducing the risk of overlooking new malware types at the expense of more frequent false alarms.

In Figure 6.14-c, we observe the results for Projection Similarity, which resembles the results for Data Augmentation but with improved performance in distinguishing between novel and known samples. This method enhances its performance even at lower confidence thresholds compared to Data Augmentation.

These results point that Ensemble Voting would be an ideal choice in identifying novel samples in the cost of lower coverage rate. Furthermore, the trade-off between coverage and risk at varied confidence levels suggests that Ensemble Voting can be calibrated to meet specific operational requirements by adjusting the threshold. For environments where the detection of novel threats is prioritized over the increased misclassification of known samples a higher threshold can be selected. Differently, in settings where false positives are a critical concern, decreasing the threshold could help mitigate this issue, in the cost of reduced sensitivity to novel malware. In summary, the Figure 6.14 highlights the adaptability to different security needs through the manipulation of the confidence threshold. Finally, we note that the scores obtained in this section for MalwareDNA are lower than the performance presented as initial results in previous section, Section 6.0.5. We believe this may be caused because of a more challenging experimental setting in this section while the initial results were done over an "easier" data, where the initial results included small number of samples and balanced data.

Class Imbalance

Next, we investigate the performance of MalwareDNA under conditions of class imbalance. The dataset setup for this investigation follows the structure outlined in the previous section, where we examined the confidence metrics. Recall that we detail the distribution of malware families in the training and testing sets in Table 2.4, and Figure 6.12. In this section, our method is compared against both supervised and semi-supervised baselines. Given our findings from previous section, we will show the results for Ensemble Voting in this section.

Model	F1	Precision	Recall
MalwareDNA (Projection Similarity)	.966 (+-.008)	.973 (+-.007)	.960 (+-.005)
MalwareDNA (Ensemble Voting)	.995 (+-.002)	.993 (+-.002)	.996 (+-.002)
MalwareDNA (Data Augmentation)	.966 (+-.012)	.971 (+-.009)	.967 (+-.008)
XGBoost	.500 (+-.005)	.468 (+-.039)	.823 (+-.013)
LightGBM	.509 (+-.006)	.460 (+-.031)	.825 (+-.019)
XGBoost-SelfTrain	.499 (+-.007)	.466 (+-.034)	.819 (+-.015)
LightGBM-SelfTrain	.510 (+-.006)	.460 (+-.031)	.824 (+-.014)

Table 6.2: Performance of MalwareDNA when classifying malware families compared to baselines.

Model	Rejection Seen	Rejection Novel
MalwareDNA (Projection Similarity)	67.16% (+- 3.38)	85.84% (+- 0.76)
MalwareDNA (Ensemble Voting)	70.11% (+- 0.40)	95.34% (+- 0.09)
MalwareDNA (Data Augmentation)	69.23% (+- 3.32)	84.67% (+- 2.76)
XGBoost	NA	NA
LightGBM	NA	NA
XGBoost-SelfTrain	11.80% (+- 1.48)	27.50% (+- 3.44)
LightGBM-SelfTrain	5.75% (+- 0.88)	13.50% (+- 2.07)

Table 6.3: Novel malware detection of MalwareDNA compared to baselines. Rejection Seen provides the false rejection predictions for the samples that belongs to known classes. Rejection Novel is the true rejection predictions for the samples that belong to a novel malware family.

At around the 30% coverage rate, MalwareDNA with Ensemble Voting achieves an F1 score of 0.995 when classifying the malware families (Table 6.2), and 95.34% true-rejection predictions for the chosen unseen family ramnit, which illustrates our method's ability to identify novel malware families (Table 6.3). Note that the relatively high rejection-seen percentage for our method, for example 70.11% for Ensemble Voting in Table 6.3, is the result of the trade-off for giving up coverage, and in return obtaining lower risk (i.e. higher performance or F1 score) and higher rate of detecting novel malware families. A user may choose the trade-off between the coverage and risk, see for example Figure 6.13 for determining the model performance for different coverage rates. In Table 6.3 it can also be seen that our baselines, including the semi-supervised ones, obtain much lower scores. The lower performance of our baselines are mainly caused by the miss-classified rare-classes as well as the novel families. For example, as seen in Table 6.3, our semi-supervised baselines with SelfTrain, while obtaining low rejection-seen percentage, they are also not able to reject much of the novel malware families and miss-classify them (XGBoost+SelfTrain only rejects 27.50% of the novel specimens). We further notice the performance degradation of our baselines for the class-imbalance problem in Figure 6.15.

Figure 6.15 shows that our baseline models yield lower performance on each of the rare malware families (emotet, fareit, and zusy), while our method maintains a higher F1 score. Note that MalwareDNA with Data Augmentation and Projection Similarity yields lower scores for the rare family emotet, while Ensemble Voting still manages to maintain its performance. Therefore,

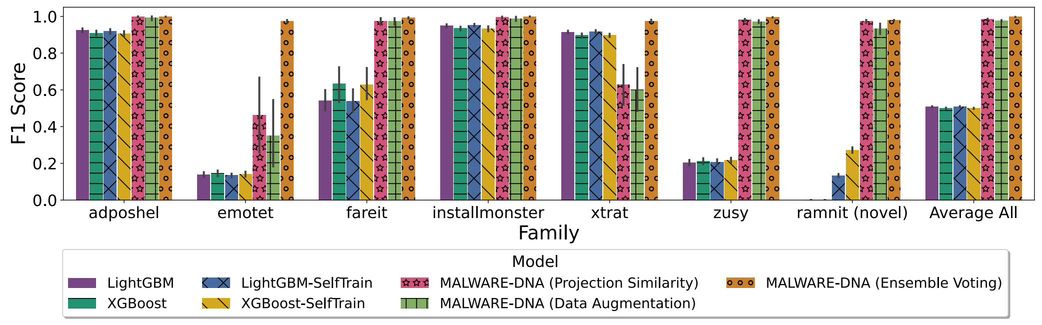


Figure 6.15: Mean F1 scores with CI is reported for each malware family when comparing MalwareDNA with different confidence metrics to both our supervised and semi-supervised baselines. It can be seen that, while our baseline's performance degrade for the rare malware families (emotet, fareit, and zusy), our method maintains its performance.

we believe that Ensemble Voting is an ideal confidence metric for handling the class imbalance problem. Our benchmarking against the baseline models and the poor performance of these models, points out both the difficulty of the task, and MalwareDNA's unique ability to both accurately classify families under class imbalance, while simultaneously detecting novel malware families.

We also demonstrate the performance of our model across each malware family, evaluated separately at various confidence thresholds for Ensemble Learning, as depicted in Figure 6.16. In this figure, the y-axis represents the average F1 score for the families included in the training set. Differently, for the novel family ramnit, the y-axis indicates the proportion of ramnit

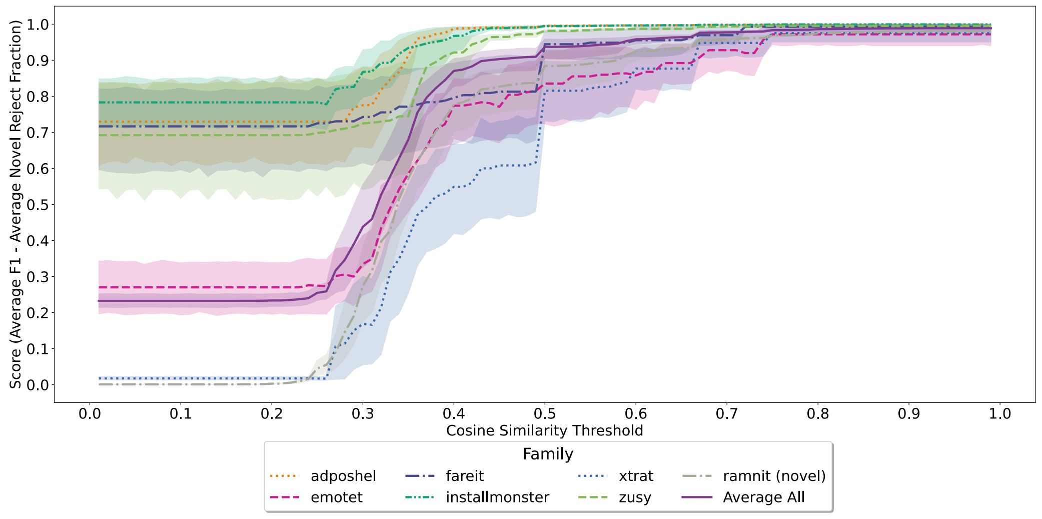


Figure 6.16: The performance of MalwareDNA, illustrated with confidence intervals at various confidence thresholds for Ensemble Voting, is presented separately for each malware family. The average F1 score for known samples and the fraction of abstaining predictions for Ramnit (novel family), are also shown. The purple solid line represents the average for all categories.

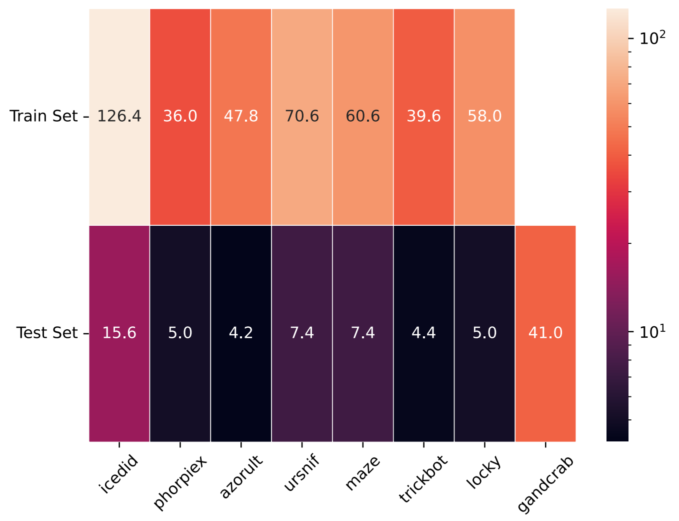
samples classified as novel using the reject option. The purple solid line represents the average performance across all families, summarizing the model's effectiveness at different confidence thresholds. A key observation from this figure is that the overall model performance begins to be optimal after the confidence threshold of 0.5. It is also notable that our model struggles the most with the malware family xtrat up to the same threshold, where the confidence interval for this family is notably wide and the average F1 score is low. Beyond a confidence threshold of 0.4, our model excels in identifying the novel ramnit specimens, correctly classifying approximately 80% of these as novel. Furthermore, beyond the confidence threshold of 0.8, the confidence intervals for all samples converge to a narrower range, indicating that the model becomes more confident in its predictions.

Experiments on the MOTIF Dataset

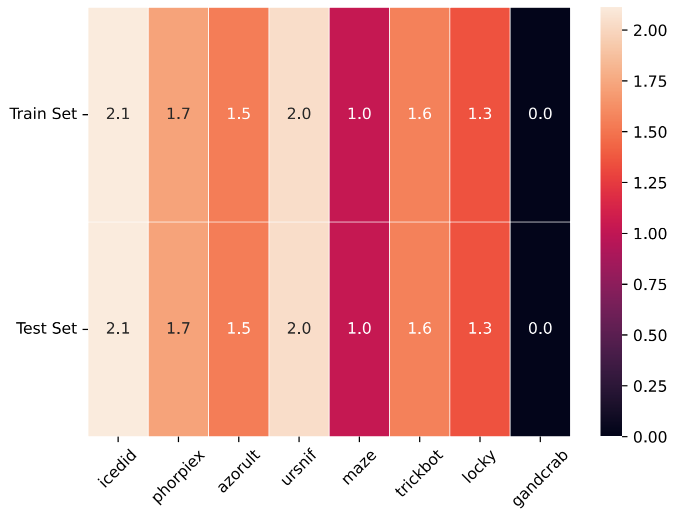
Family Label	Aliases
181	**icedid**
285	**phorpiex**, trik
30	**azorult**, rultazo
416	**ursnif**, gozi, goziisfb, isfb, dreambot, hpursnif, wastenif
230	**maze**, mazeransom, mazedec, chacha
407	**trickbot**, trickster, totbrick, trickybot, trick, hptrickbot, trickbotmodule, trickbt
217	**locky**, lockycrypt, hplocky
150	**gandcrab**, gandcrypt, gandcrab4, ransomgandcrab, gandcrabv4, grandcrab, hpgandcrab, gandcrab04

Table 6.4: The table presents a comprehensive list of aliases for the families included in this dataset, alongside the original family labels from the MOTIF dataset. The selected first alias that will be shown in our experiments are highlighted for each family.

We next advance to demonstrating the model's capabilities with another dataset, specifically the MOTIF dataset. Given that the MOTIF dataset is considerably smaller than EMBER-2018, we have included an additional malware family in our experiments, increasing the total number of families to eight. The distribution of the training and testing sets for this phase of experiments is shown in Figure 6.17. Unlike the EMBER-2018 dataset, which provides AVClass labels for malware families, the MOTIF dataset includes aliases for these families. Since each antivirus engine may assign different



(a)



(b)

Figure 6.17: Dataset distribution for the experiments for MOTIF dataset is shown. The x axis displays the first alias provided for the given family in the MOTIF dataset. **(a)** The heatmap displays the average number of samples in the train and test sets for each malware family. **(b)** For each of the 10 experiments, we randomly sample the dataset. This heatmap illustrates the confidence intervals for the samples in the train/test split for each malware family.

aliases to the malware families, each sample may contain multiple aliases. In Figure 6.17, we display the first alias that appears for each given sample in the dataset. We have chosen the malware family represented by the first alias, gandcrab, to represent the novel family, and have excluded it from the training set.

For completeness, Table 6.4 displays the full list of aliases associated with each malware family. This table also includes the original malware family labels from the dataset. The aliases that are utilized in our figures and results are highlighted in bold.

We first compare our method against semi-supervised baselines using the risk-coverage curve and the area under these curves as shown in Figure 6.18. Due to the sub-optimal performance of the confidence metric with Data Augmentation observed in our previous experiments, we now limit our analysis to using Ensemble Voting and Projection Similarity as confidence metrics for MalwareDNA. The initial observation from this figure, consistent with previous results, is that our method surpasses the semi-supervised baselines, which are not tailored to detect novel malware families. Additionally, both the baseline models and our method exhibit a performance decline on this dataset, indicating that MOTIF dataset is more challenging than EMBER-

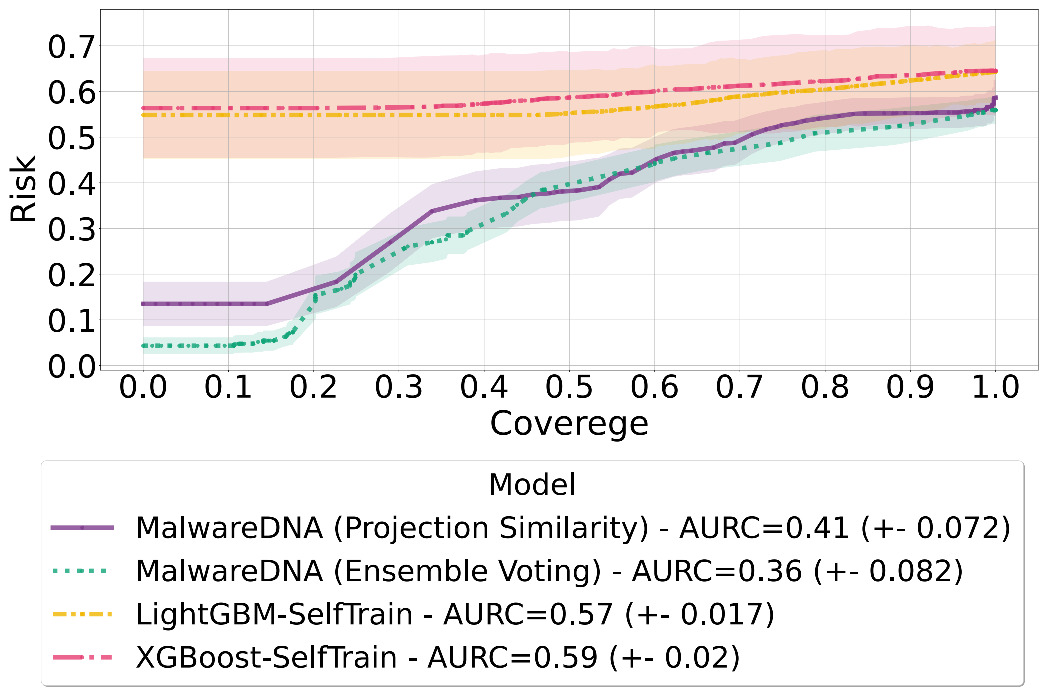


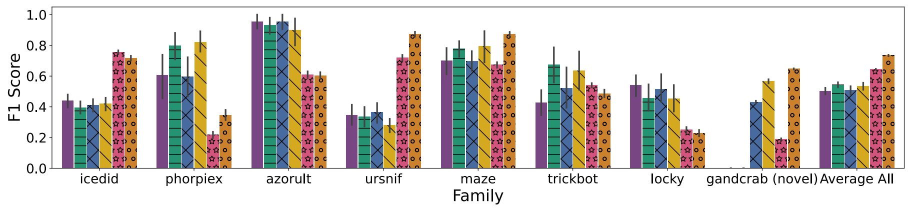
Figure 6.18: Risk-Coverage (RC) curve when classifying malware families and novel malware together with the area under the RC (AURC) for different MalwareDNA confidence metrics and our semi-supervised baselines on the MOTIF dataset.

2018. In the EMBER-2018 dataset, we assessed the confidence metrics using a risk-coverage curve in Section 6.0.5, where the MalwareDNA with Ensemble Voting achieved an AURC score of 0.16. However, while this metric continues to perform best in the MOTIF dataset, the score has decreased to 0.36. By contrast, our baseline model attains a score of 0.57, significantly lower than our approach. Another notable finding from this figure is that Ensemble Voting maintains approximately 0 risk until a coverage rate of about 0.15.

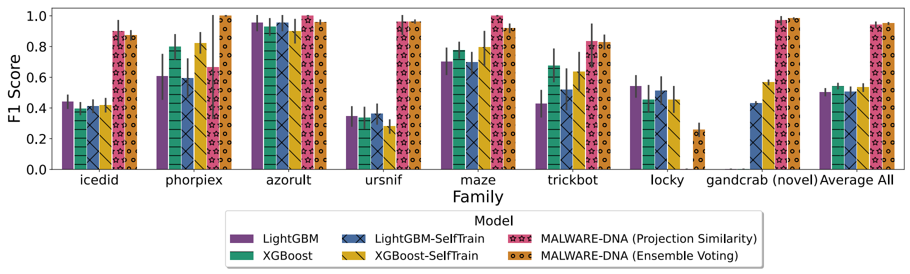
Therefore, if coverage is not the primary objective in a malware detection system, our system could be effectively employed to classify malware families with high accuracy at lower coverage levels.

We next examine the performance of the models at low and high coverage, as indicated by F1 scores presented in Figure 6.19. The first point to note is that our model's performance deteriorates at high coverage, achieving an F1 score of approximately 0.6, as shown in Figure 6.19-a, yet it still surpasses our baselines. This figure also illustrates that our semi-supervised baselines outperform MalwareDNA with Projection Similarity, while Ensemble Voting remains the most effective for detecting novel specimens. Conversely, in Figure 6.19-b, where we reduced the coverage to enhance performance, our method's F1 score approaches 0.95, significantly improving the detection of novel malware families. However, this improvement comes with the cost of a higher number of undecidable cases among the malware samples, resulting in a low coverage rate. In addition, at lower coverage, a notable issue arises. Our method fails to accurately classify the locky malware family using Projection Similarity.

We further compare the models in Tables 6.5 and 6.6. Table 6.5 displays results for low coverage, which is tailored towards detecting more novel



(a)



(b)

Figure 6.19: Mean F1 scores, accompanied by confidence intervals (CIs), are reported for each malware family when comparing MalwareDNA across various confidence metrics with both our supervised and semi-supervised baselines. We assess the performance of all models at both low and high coverage rates. Recall that the coverage rate refers to the proportion of non-abstaining predictions, indicating instances where the model has made a decision. A high coverage rate implies that the model made decisions for the majority of samples, whereas a low coverage rate increases the frequency of abstaining predictions to gain model performance. Low coverage results are selected and reported based on the average across all experiments. Differently, high coverage results are determined by selecting confidence levels that yield the most accurate detection of a novel malware family as novel using the reject-option. **(a)** High coverage rate results. **(b)** Low coverage rate results.

Model	**F1**	**Precision**	**Recall**	**Coverage (%)**	**Reject Novel (%)**
MalwareDNA (Projection Similarity)	.858 (+-.059)	.912 (+-.058)	.884 (+-.060)	13.17 (+-1.16)	96.81 (+-2.29)
MalwareDNA (Ensemble Voting)	.906 (+-.012)	.922 (+-.013)	.907 (+-.013)	16.93 (+-0.51)	97.94 (+-0.21)
XGBoost	.562 (+-.113)	.542 (+-.125)	.734 (+-.125)	100 (+-0)	0 (+-0)
LightGBM	.545 (+-.111)	.502 (+-.121)	.745 (+-.130)	100 (+-0)	0 (+-0)
XGBoost-SelfTrain	.499 (+-.007)	.569 (+-.029)	.682 (+-.025)	72.85 (+-2.98)	45.35 (+-4.87)
LightGBM-SelfTrain	.324 (+-.046)	.440 (+-.020)	.598 (+-.019)	27.86 (+-1.90)	27.86 (+-3.71)

Table 6.5: F1, Precision, Recall, and detection percentage for novel family scores shown for our method as compared to our baselines for the confidence metrics that resulted in **low coverage**.

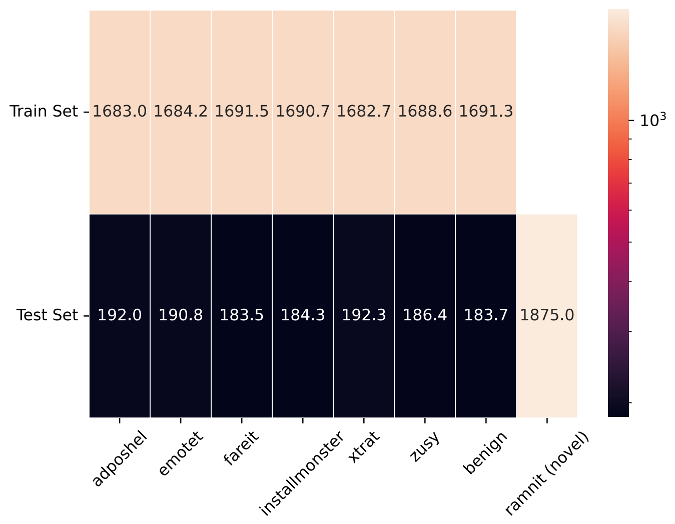
Model	**F1**	**Precision**	**Recall**	**Coverage (%)**	**Reject Novel (%)**
MalwareDNA (Projection Similarity)	.432 (+-.008)	.565 (+-.010)	.445 (+-.008)	17.37 (+-0.79)	96.81 (+-2.29)
MalwareDNA (Ensemble Voting)	.634 (+-.010)	.683 (+-.010)	.646 (+-.011)	52.91 (+-1.32)	56.72 (+-1.48)
XGBoost	.562 (+-.113)	.542 (+-.125)	.734 (+-.125)	100 (+-0)	0 (+-0)
LightGBM	.545 (+-.111)	.502 (+-.121)	.745 (+-.130)	100 (+-0)	0 (+-0)
XGBoost-SelfTrain	.499 (+-.046)	.569 (+-.030)	.682 (+-.024)	72.85 (+-2.91)	45.35 (+-4.87)
LightGBM-SelfTrain	.324 (+-.039)	.440 (+-.020)	.598 (+-.018)	85.14 (+-2.04)	27.86 (+-3.81)

Table 6.6: F1, Precision, Recall, and detection percentage for novel family scores shown for our method as compared to our baselines for the confidence metrics that resulted in **high coverage**.

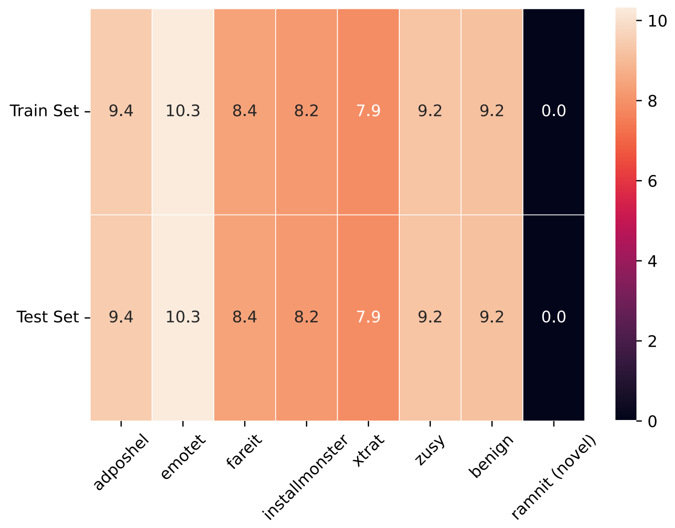
families and maintaining higher model performance. Conversely, Table 6.6 presents the results for high coverage. It can be seen in these tables that our model achieves a high F1 score of 0.9 with Ensemble Voting at a low coverage rate of 16%. At this coverage rate, 97% of novel families are correctly classified as novel using the reject-option. In contrast, our baselines perform considerably worse, achieving a higher recall but a lower precision score. This disparity indicates that many classes are detected inaccurately, stemming also from the inclusion of samples from the novel class. At a higher coverage, the performance of MalwareDNA with Ensemble Voting declines to an F1 score of 0.63, yet it still surpasses our baselines at an increased coverage rate of 52%. It is important to note that the best-performing baseline model achieves an F1 score of 0.56 without the reject-option, and thus this score reflects a 100% coverage rate. Another notable observation is that Projection Similarity does not significantly increase its coverage, remaining at around 17%, but it still detects 96% of the novel families effectively. The main takeaway from these tables is that our model can be used to classify malware families more accurately than our baselines; however, our baselines is be able to obtain a higher coverage rate.

Simultaneous Classification of Malware, Malware Families, and Novel Malware

As the final approach to analyzing the performance of MalwareDNA, we have designed an experiment that incorporates benign samples into the dataset. For this purpose, we return to the EMBER-2018 dataset. We include a figure that depicts the training and testing splits, as shown in Figure 6.20. This experimental setup aims to evaluate our model's ability to simultaneously distinguish malware from benign-ware, classify malware families, and detect novel specimens. While our previous analyses focused on specific experimental conditions, testing targeted settings where the specimen was



(a)



(b)

Figure 6.20: Dataset distribution for the experiments for "Simultaneous Classification of Malware, Malware Families, and Novel Malware" is shown. **(a)** The heatmap displays the average number of samples in the train and test sets for each malware family. **(b)** For each of the 10 experiments, we randomly sample the dataset. This heatmap illustrates the confidence intervals for the samples in the train/test split for each malware family.

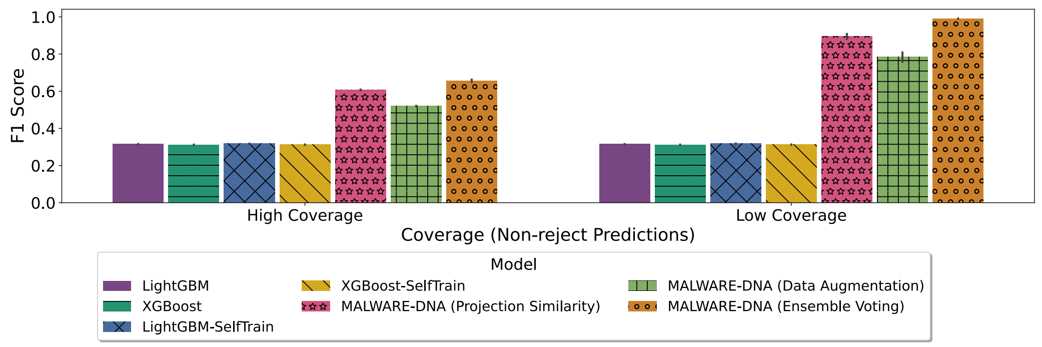


Figure 6.21: This figure presents the average F1 scores for each supervised and semi-supervised model compared to MalwareDNA, utilizing various confidence metrics. The performance, represented by the average F1 score across all malware families and benign specimens, is reported for both high and low coverage rates.

already known to be malware, this setup emulates a more realistic scenario where a specimen could also be benign.

We report our results in Figure 6.21, where we compare the methods at low and high coverage rates. Low coverage is reported using the average of all results, while high coverage is determined by selecting the confidence score that achieves the highest detection rate of novel malware families. Notably, our model outperforms the baselines at both high and low coverage rates across all confidence metrics. The scores for baselines remain constant at different coverage rates as they lack a reject-option. At higher coverage levels, our model's performance again declines for all confidence methods, yet it still exceeds that of the baselines. Although previous work reported state-of-the-art results with these baselines, their lower scores in our experiments can be attributed to the more challenging task of distinguishing between benign-ware and malware, classifying malware families, and detecting novel specimens all at the same time.

Model	**High Coverage (%)**	**Low Coverage (%)**
MalwareDNA (Projection Similarity)	88.11 (+-.509)	30.88 (+-1.985)
MalwareDNA (Ensemble Voting)	57.56 (+-1.167)	16.57 (+-.461)
MalwareDNA (Data Augmentation)	94.75 (+-.376)	36.68 (+-2.721)
XGBoost	100 (+-0)	100 (+-0)
LightGBM	100 (+-0)	100 (+-0)
XGBoost-SelfTrain	90.65 (+-1.555)	90.65 (+-1.419)
LightGBM-SelfTrain	96.26 (+-.648)	95.98 (+-.649)

Table 6.7: Coverage rates at high and low experiment settings are reported for each model.

In Table 6.7, we present the actual coverage rates for the "low" and "high" coverage categories as outlined in Figure 6.21. The high coverage setting for Ensemble Voting is 57%, while the low coverage setting decreases to 16%. Projection Similarity and Data Augmentation achieve significantly higher coverage rates, reaching 88% and 94% respectively for the high coverage settings, while their rates decrease to around 30% in the low coverage settings. These results demonstrate that with a high coverage rate of approximately 94% for Data Augmentation, we can exceed the performance of our baseline models, achieving an F1 score of around 0.55.

Finally, Table 6.8 presents the average tuning, training, and testing times

Model	**Tune Time**	**Train Time**	**Test Time**
MalwareDNA (Projection Similarity)	0	19916.18 (+-3021.74)	217.3 (+-35.01)
MalwareDNA (Ensemble Voting)	0	19916.18 (+-3021.74)	214.4 (+-33.70)
MalwareDNA (Data Augmentation)	0	19916.18 (+-3021.74)	1046.8 (+-187.07)
XGBoost	398.59 (+-55.63)	1.40 (+-0.32)	0.03 (+-0.01)
LightGBM	769.69 (+-67.50)	8.44 (+-1.28)	0.23 (+-0.03)
XGBoost-SelfTrain	398.59 (+-55.63)	14.25 (+-3.99)	0.08 (+-0.01)
LightGBM-SelfTrain	769.69 (+-67.50)	41.51 (+-13.24)	0.10 (+-0.02)

Table 6.8: This table shows the tuning, training, and testing time for each methods. Time is reported with seconds and confidence interval is included.

for all models. The first point to make on this table is that our method does not require a tuning phase, as it employs dimensionality reduction with automatic model determination for signature extraction, which does not require hyper-parameter selection. In contrast, our baseline models undergo extensive tuning with Optuna across 200 trials, utilizing stratified sampling across five cross-validations in each trial to identify the optimal hyper-parameters. It is important to note that the tuning times for the supervised and semi-supervised models are identical, as the semi-supervised models use the same hyper-parameters as their supervised counterparts. In addition, this tuning time for the baseline models can be reduced by distributing the Optuna trials on a HPC system.

The training time for MalwareDNA is significantly longer than that of our baseline models. This duration refers to the time spent building the latent signature archive, facilitated by the use of four HPC compute nodes.

Potential reductions in training time could be explored in future work, possibly through the utilization of a greater number of compute nodes or more efficient automatic model determination techniques, which are discussed in the Future Work section, Section 9.

Lastly, the testing or prediction time for our model is also considerably longer as compared to our baseline models. Future improvements include speeding up this process by employing multi-processing or vector multiplication techniques to compute cosine similarity scores across the samples in the testing set.

Chapter 7

Open Source Code Outcomes

Software	Link	Dense	Sparse	GPU	CPU	Multiprocessing	HPC
RFoT	`https://github.com/MaksimEkin/RFoT`	✓	✓	✓	✓	✓	—
pyCP_ALS	`https://github.com/MaksimEkin/pyCP_ALS`	—	✓	—	✓	—	—
pyCP_APR	`https://github.com/lanl/pyCP_APR`	✓	✓	✓	✓	—	—
T-ELF	`https://github.com/lanl/T-ELF`	✓	✓	✓	✓	✓	✓

Table 7.1: Software packages released through the studies conducted on tensor decomposition, including those presented in this dissertation, are listed below. The **Software** column provides the name of each software package. The **Dense** and **Sparse** columns indicate whether the software can handle dense and sparse tensors and matrices, respectively. The **GPU** and **CPU** columns specify the software's compatibility with GPU and CPU hardware. Finally, the **HPC** column denotes the software's high-performance computing capabilities.

We have developed and released several Python libraries, each with unique capabilities. Table 7.1 provides a summary of these libraries. Notably, the RFoT package is available with multiprocessing features, supporting both GPU for accelerated computing and CPU for standard computing. This library efficiently handles both sparse and dense matrices and tensors.

The pyCP_APR library was initially developed for cyber anomaly detection. RFoT utilizes this library for tensor decomposition and for identifying malware clusters. Additionally, we introduced pyCP_ALS as a separate library, which RFoT also uses for tensor decomposition-based clustering. Importantly, pyCP_APR includes GPU capabilities, which significantly enhance its computation speed compared to its predecessor available in Matlab Tensor Toolbox [22, 72]. We have previously demonstrated this performance gain with use of GPUs for CP-APR algorithm in [72].

HNMFk Classifier and MalwareDNA codes have been patented, and thus, we have not released these codes publicly. However, both techniques employ the hierarchical NMFk feature from T-ELF, which is equipped with advanced High-Performance Computing (HPC) capabilities which we have discussed previously in Section 6.0.4.

RFoT, pyCP_ALS, and pyCP_APR employ a custom sparse tensor format, which was originally introduced in the MATLAB Tensor Toolbox [21]. This format enables efficient representation of tensors and matrices in COO format. In this structure, a matrix records the coordinates of non-zero elements—each row corresponding to a single coordinate—and a separate vector stores the non-zero elements themselves. The COO representation allow for efficient dimensionality reduction operations on sparse tensors and matrices. Differently, T-ELF utilizes the CuPy library, specifically its CuPyx extension, to facilitate efficient sparse matrix operations on both CPUs and GPUs [136]. As discussed in Section 6.0.4, T-ELF leverages threading to manage concurrent operations and to launch parallel tasks, whereas RFoT harnesses Python's Joblib library for multiprocessing [102]. The shift from Joblib was strategically made to enhance the efficiency of GPU operations with CuPy during parallel task execution. Moreover, while pyCP_ALS and pyCP_APR have been translated into Python with additional GPU capabilities via the PyTorch [138] backend, they preserve the core design principles of the original MATLAB implementation from the MATLAB Tensor Toolbox. Differently, T-ELF [69] incorporates several distinct algorithmic design modifications, including improved GPU host-to-device communication, optimized data transfer and copying, and enhanced settings for efficient process utilization, among others.

Finally, all the software listed in Table 7.1 follows to Python standards and is modularized for code organization. Each tool is designed as an installable Python library, which can be installed using pip [1]. These libraries are also accompanied by example Jupyter Notebooks to demonstrate their use [110] and include unit tests that integrate with GitHub's Continuous Integration (CI) pipeline. The documentation for each library is thorough, with a documentation website generated using Sphinx and hosted as a GitHub Page [159]. Additionally, T-ELF utilizes the Git software development life cycle, complete with a project board. These projects also feature easy installation guides that leverage Anaconda's Conda environment for setup [5], facilitating the use of our tools across various devices. Moreover, all libraries are designed with an API style following the Scikit-learn's design principles, which fosters a user-friendly and familiar interface for interacting with the packages [140]. Overall, our libraries are designed to be efficient, user-friendly, and follow rigorous software development standards.

Chapter 8

Conclusions

In this dissertation, we presented three novel semi-supervised malware characterization methods based on tensor decomposition and machine learning. Our approaches, named RFoT, HNMFk Classifier, and MalwareDNA, are designed to address various shortcomings in the large-scale malware analysis domain. We have demonstrated the efficacy of our methods using two datasets for Windows PE malware and benign-ware, using static malware analysis based features. Our experiments were conducted in real-world-like settings to test the methods against extreme class imbalance, large-scale data, and up to extreme number of malware families. These tests also included scenarios with limited labeled training data, the introduction of previously unseen malware families, and the simultaneous analysis of benign software and malware.

Additionally, we conducted ablation studies and hyper-parameter analyses to validate our hypotheses. We benchmarked our results against other supervised and semi-supervised techniques that have previously reported state-of-the-art outcomes on these datasets. Our findings reveal that while baseline models lose performance under more realistic experimental conditions, our methods significantly outperform them with a trade-off in reduced coverage rate. A key feature of our approaches is the incorporation of a reject-option, which allows the model to express uncertainty, or the ability to say "I do not know," instead of confidently making a wrong decision. In the domain of large-scale malware analysis, the reject-option feature allow the detection new threats, particularly novel malware, and helps maintain model performance in environments with scarce labeled data. By enabling the model to reduce its coverage rate, this feature effectively minimizes the risk of misclassification, thus providing a more robust and reliable analysis in uncertain or evolving threat landscapes.

In operational settings, security analysts may consider using our methods, which yield results with lower risk (higher accuracy), as the first layer of defense. The abstained predictions can be analyzed in a second layer using traditional machine learning approaches, such as an ensemble of XGBoost and LightGBM models.

Furthermore, we have published several open-source libraries to support our methods, enhancing their accessibility and functionality with features such as multiprocessing, high-performance computing (HPC), and GPU-accelerated computation.

Chapter 9

Future Work

Future work will include testing our models on additional datasets. For instance, the recently released MalDICT dataset, which includes classification tasks based on behavior, malware family, and vulnerabilities [104], will be utilized. Future experiments will explore whether novel vulnerabilities or behavioral classes of malware can be detected via reject-option using the MalDICT dataset. Furthermore, future studies could employ a temporal train-test split of this dataset to implement a time-based data division to support the idea behind real-world like settings. Additional future work will extend these experiments to incorporate features derived from dynamic malware analysis.

Future work will also involve strategies to enhance the efficiency of MalwareDNA. In Section 6.0.5, we demonstrated that the computational time required for MalwareDNA is greater than that for our baselines. This computational time could be reduced by utilizing existing HPC capabilities and by deploying a larger number of compute nodes during the construction of the signature archive. Additionally, prediction times could be minimized by employing vector and matrix operations for the calculation of similarity or confidence scores.

Furthermore, the time required to construct the latent signature archive in MalwareDNA and to perform hierarchical clustering in the HNMFk Classifier could be reduced through a more efficient automatic model determination technique. Recall that both MalwareDNA and HNMFk Classifier employ NMFk for dimensionality reduction, as discussed in Section 2.0.5 [10]. NMFk heuristically searches for the optimal matrix/tensor rank, or K, by decomposing the matrix/tensor linearly for each rank within a specified range. For example, given `Ks=[1,2,3]`, NMFk will decompose the matrix $\mathbf{X}$ for each $k = 1$, $k = 2$, and $k = 3$. It then evaluates the stability and accuracy for each rank to determine the ideal rank. The selection of this rank is based on a marked decline in silhouette scores from the clustering of latent patterns and cluster centroids. This specifically involves the clustering of bootstrap samples of **W** and **H** latent factors, for a given rank, followed by a convergence in the relative reconstruction error. This technique, a LANL patented method for automatic model determination, was developed by Dr. Alexandrov [11]. The necessity of decomposing the matrix for each rank can lead to prolonged computation times, especially when NMFk operations are conducted thousands of times in a hierarchical framework. Future work include integrating heuristics for more efficient search of matrix rank into HNMFk Classifier and MalwareDNA [26].

References

[1] Python package index - pypi. URL https://pypi.org/.

[2] Cost of a data breach report. Technical report, IBM, 2019. URL https://www.accenture.com/%5Facnmedia/PDF-96/Accenture-2019-Cost-of-Cybercrime-Study-Final.pdf.

[3] State of malware report. Technical report, Malwarebytes Labs, February 2020. URL https://www.accenture.com/%5Facnmedia/PDF-96/Accenture-2019-Cost-of-Cybercrime-Study-Final.pdf.

[4] Data breach investigations report 2021. Technical report, Verizon, 2021. URL https://enterprise.verizon.com/resources/reports/dbir/.

[5] Anaconda software distribution, 2024. URL https://docs.anaconda.com/.

[6] Salma Abdelmonem, Shahd Seddik, Rania El-Sayed, and Ahmed S. Kaseb. Enhancing image-based malware classification using semi-supervised learning. In *2021 3rd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, pages 125–128, 2021. doi: 10.1109/NILES53778.2021.9600511.

[7] Amina Adadi. A survey on data-efficient algorithms in big data era. *Journal of Big Data*, 8(1):24, 2021.

[8] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the sixth ACM conference on data and application security and privacy*, pages 183–194, 2016.

[9] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[10] Boian Alexandrov, Velimir Vesselinov, and Kim Orskov Rasmussen.

Smarttensors unsupervised ai platform for big-data analytics. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2021. URL `https://www.lanl.gov/collaboration/smart-tensors/`. LA-UR-21-25064.

[11] BS. Alexandrov, LB. Alexandrov, and VG. Stanev et al. Source identification by non-negative matrix factorization combined with semi-supervised clustering. *US Patent S10,776,718*, 2020.

[12] Ludmil B Alexandrov, Serena Nik-Zainal, David C Wedge, Samuel AJR Aparicio, Sam Behjati, Andrew V Biankin, Graham R Bignell, Niccolo Bolli, Ake Borg, Anne-Lise Børresen-Dale, et al. Signatures of mutational processes in human cancer. *Nature*, 500(7463):415–421, 2013.

[13] Ludmil B Alexandrov, Serena Nik-Zainal, David C Wedge, Peter J Campbell, and Michael R Stratton. Deciphering signatures of mutational processes operative in human cancer. *Cell reports*, 3(1):246–259, 2013.

[14] Ludmil B Alexandrov, Jaegil Kim, Nicholas J Haradhvala, Mi Ni Huang, Alvin Wei Tian Ng, Yang Wu, Arnoud Boot, Kyle R Covington, Dmitry A Gordenin, Erik N Bergstrom, et al. The repertoire

of mutational signatures in human cancer. *Nature*, 578(7793):94–101, 2020.

[15] H. Anderson and P. Roth. Ember: An open dataset for training static pe malware machine learning models. *ArXiv*, abs/1804.04637, 2018.

[16] H. S. Anderson and P. Roth. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints*, April 2018.

[17] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[18] Venkata Atluri. Malware classification of portable executables using tree-based ensemble machine learning. In *2019 SoutheastCon*, pages 1–6. IEEE, 2019.

[19] Nureni Ayofe Azeez, Oluwanifise Ebunoluwa Odufuwa, Sanjay Misra, Jonathan Oluranti, and Robertas Damaševičius. Windows pe malware detection using ensemble learning. In *Informatics*, volume 8, page 10. Multidisciplinary Digital Publishing Institute, 2021.

[20] Brett W. Bader and Tamara G. Kolda. Algorithm 862: MATLAB tensor classes for fast algorithm prototyping. *ACM Transactions on Mathematical Software*, 32(4):635–653, December 2006. doi: 10.1145/1186785.1186794.

[21] Brett W. Bader and Tamara G. Kolda. Efficient matlab computations with sparse and factored tensors. *SIAM Journal on Scientific Computing*, 30(1):205–231, 2008. doi: 10.1137/060676489. URL `https://doi.org/10.1137/060676489`.

[22] Brett W. Bader, Tamara G. Kolda, et al. Matlab tensor toolbox version 2.6. Available online, February 2015. URL `http://www.sandia.gov/~tgkolda/TensorToolbox/`.

[23] Yuval Bahat and Gregory Shakhnarovich. Confidence from invariance to image transformations. *arXiv preprint arXiv:1804.00657*, 2018.

[24] Yuval Bahat and Gregory Shakhnarovich. Classification confidence estimation with test-time data-augmentation. *arXiv preprint arXiv:2006.16705*, 2020.

[25] Márton Bak, Dorottya Papp, Csongor Tamás, and Levente Buttyán. Clustering iot malware based on binary similarity. In *NOMS 2020-2020*

IEEE/IFIP Network Operations and Management Symposium, pages 1–6. IEEE, 2020.

[26] Ryan Barron. Binary bleed k search. Manuscript in preparation, 2024.

[27] Ryan Barron, Maksim E. Eren, Manish Bhattarai, Selma Wanna, Nicholas Solovyev, Kim Rasmussen, Boian S. Alexandrov, Charles Nicholas, and Cynthia Matuszek. Cyber-security knowledge graph generation by hierarchical nonnegative matrix factorization. In *2024 12th International Symposium on Digital Forensics and Security (ISDFS)*, pages 1–6, 2024. doi: 10.1109/ISDFS60797.2024.10527237.

[28] Casey Battaglino, G. Ballard, and T. Kolda. A practical randomized CP tensor decomposition. *SIAM J. Matrix Anal. Appl.*, 39:876–901, 2018.

[29] P. Bhandary, C. Vieson, A. Kiendrebeogo, I. Adetunji, R. Joyce, M. E. Eren, and C. Nicholas. Malware antivirus scan pattern mining via tensor decomposition, 2022. Presented at the 13th Annual Malware Technical Exchange Meeting, Online, 2022.

[30] Christopher M Bishop. Bayesian pca. *Advances in neural information processing systems*, pages 382–388, 1999.

[31] K. Bissell and L. Ponemon. The cost of cybercrime. Technical report, Accenture, Ponemon Institute, 2019. URL `https://www.accenture.com/%5Facnmedia/PDF-96/Accenture-2019-Cost-of-Cybercrime-Study-Final.pdf`.

[32] Ismael Boureima, Manish Bhattarai, Maksim E. Eren, Nick Solovyev, Hristo Djidjev, and Boian S. Alexandrov. Distributed out-of-memory svd on cpu/gpu architectures. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8, 2022. doi: 10.1109/HPEC55821.2022.9926288.

[33] Ismael Boureima, Manish Bhattarai, Maksim Eren, Erik Skau, Philip Romero, Stephan Eidenbenz, and Boian Alexandrov. Distributed out-of-memory nmf on cpu/gpu architectures. 2023.

[34] Ismael Boureima, Manish Bhattarai, Maksim Eren, Erik Skau, Philip Romero, Stephan Eidenbenz, and Boian Alexandrov. Distributed out-of-memory nmf on cpu/gpu architectures. *The Journal of Supercomputing*, pages 3970–3999, 2024.

[35] Rasmus Bro and Sijmen De Jong. A fast non-negativity-constrained

least squares algorithm. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 11(5):393–401, 1997.

[36] Jean-Philippe Brunet, Pablo Tamayo, Todd R Golub, and Jill P Mesirov. Metagenes and molecular pattern discovery using matrix factorization. *Proceedings of the national academy of sciences*, 101(12): 4164–4169, 2004.

[37] David Bruns-Smith, Muthu Manikandan Baskaran, James R. Ezick, Thomas Henretty, and Richard A. Lethin. Cyber security through multidimensional data decompositions. *2016 Cybersecurity Symposium (CYBERSEC)*, pages 59–67, 2016.

[38] John Canny. Gap: a factor model for discrete data. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 122–129, 2004.

[39] Léna Carel and Pierre Alquier. Simultaneous dimension reduction and clustering via the nmf-em algorithm. *Advances in Data Analysis and Classification*, 15:231–260, 2021.

[40] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019.

[41] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. URL `http://doi.acm.org/10.1145/2939672.2939785`.

[42] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4, 2015.

[43] Eric C. Chi and Tamara G. Kolda. On tensors, sparsity, and non-negative factorizations. *SIAM J. Matrix Anal. Appl.*, 33:1272–1299, 2012.

[44] Yo Joong Choe, Aditya Gangrade, and Aaditya Ramdas. Counterfactually comparing abstaining classifiers. *arXiv preprint arXiv:2305.10564*, 2023.

[45] Ilay Cordonsky, Ishai Rosenberg, Guillaume Sicard, and Eli Omid David. Deeporigin: End-to-end deep learning for detection of new mal-

ware families. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2018. doi: 10.1109/IJCNN.2018.8489667.

[46] George E. Dahl, Jack W. Stokes, Li Deng, and Dong Yu. Large-scale malware classification using random projections and neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3422–3426, 2013. doi: 10.1109/ICASSP.2013.6638293.

[47] Lisandro Dalcin and Yao-Lung L. Fang. mpi4py: Status update after 12 years of development. *Computing in Science & Engineering*, 23(4): 47–54, 2021. doi: 10.1109/MCSE.2021.3083216.

[48] Lisandro D. Dalcin, Rodrigo R. Paz, Pablo A. Kler, and Alejandro Cosimo. Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124–1139, 2011. ISSN 0309-1708. doi: https://doi.org/10.1016/j.advwatres.2011.04.013. URL `https://www.sciencedirect.com/science/article/pii/S0309170811000777`. New Computational Methods and Software Tools.

[49] Lisandro Dalcín, Rodrigo Paz, and Mario Storti. Mpi for python. *Journal of Parallel and Distributed Computing*, 65(9):1108–1115,

2005. ISSN 0743-7315. doi: https://doi.org/10.1016/j.jpdc.2005.03.010. URL `https://www.sciencedirect.com/science/article/pii/S0743731505000560`.

[50] Lisandro Dalcín, Rodrigo Paz, Mario Storti, and Jorge D'Elía. Mpi for python: Performance improvements and mpi-2 extensions. *Journal of Parallel and Distributed Computing*, 68(5):655–662, 2008. ISSN 0743-7315. doi: https://doi.org/10.1016/j.jpdc.2007.09.005. URL `https://www.sciencedirect.com/science/article/pii/S0743731507001712`.

[51] Thiago de Paulo Faleiros and Alneu de Andrade Lopes. On the equivalence between algorithms for non-negative matrix factorization and latent dirichlet allocation. In *ESANN*, 2016.

[52] Y. Ding, J. Liu, J. Xiong, and Y. Shi. Revisiting the evaluation of uncertainty estimation and its application to explore model complexity-uncertainty trade-off. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 22–31, Los Alamitos, CA, USA, jun 2020. IEEE Computer So-

ciety. doi: 10.1109/CVPRW50498.2020.00010. URL https://doi.ieeecomputersociety.org/10.1109/CVPRW50498.2020.00010.

[53] Yukun Ding, Jinglan Liu, Jinjun Xiong, and Yiyu Shi. Revisiting the evaluation of uncertainty estimation and its application to explore model complexity-uncertainty trade-off. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 4–5, 2020.

[54] Ran El-Yaniv et al. On the foundations of noise-free selective classification. *Journal of Machine Learning Research*, 11(5), 2010.

[55] M. E. Eren. pyCP_ALS. https://github.com/MaksimEkin/pyCP_ALS, 2022.

[56] M. E. Eren. Random forest of tensors (rfot) master's thesis, 2022. Master's Thesis in Computer Science at the University of Maryland, Baltimore County Department of Computer Science and Electrical Engineering. 2022.

[57] M. E. Eren, J. S. Moore, and B. S. Alexandrov. Multi-dimensional anomalous entity detection via poisson tensor factorization. In *2020*

IEEE International Conference on Intelligence and Security Informatics (ISI), pages 1–6, 2020. doi: 10.1109/ISI49825.2020.9280524.

[58] M. E. Eren, J. S. Moore, and B. S. Alexandrov. Multi-dimensional anomalous entity detection via poisson tensor factorization. In *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 1–6, 2020. doi: 10.1109/ISI49825.2020.9280524.

[59] M. E. Eren, M. Bhattarai, R. J. Joyce, E. Raff, C. Nicholas, and B. Alexandrov. Semi-supervised classification of malware families via hierarchical non-negative matrix factorization with automatic model determination. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2021. LA-UR-21-29919.

[60] M. E. Eren, M. Bhattarai, R. J. Joyce, E. Raff, C. Nicholas, and B. S. Alexandrov. Semi-supervised classification of malware families via hierarchical non-negative matrix factorization with automatic model determination. In *ACM Transactions on Privacy and Security (TOPS)*, 2021.

[61] M. E. Eren, J. S. Moore, E. Skau, M. Bhattarai, G. Chennupati, and

B. S. Alexandrov. pycp_apr. `https://github.com/lanl/pyCP_APR`, 2021.

[62] M. E. Eren, C. Nicholas, R. McDonald, and C. Hamer. Random forest of tensors (rfot), 2021. Presented at the 12th Annual Malware Technical Exchange Meeting, Online, 2021.

[63] M. E. Eren, C. Nicholas, E. Raff, R. Yus, J. S. Moore, and B. S. Alexandrov. Rfot. `https://github.com/MaksimEkin/RFoT`, 2022.

[64] M. E. Eren, B. S. Alexandrov, Bhattarai, K. Orskov, and C. K. Nicholas. Data identification and classification method, apparatus, and system. *US Proviosional Patent S167632.000, 63/472,188, Triad National Security, LLC*, 2023.

[65] M. E. Eren, M. Bhattarai, K. Rasmussen, B. S. Alexandrov, and C. Nicholas. Malwaredna: Simultaneous classification of malware, malware families, and novel malware. In *2023 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 1–3, 2023.

[66] M. E. Eren, K. Rasmussen, C. Nicholas, and B. S. Alexandrov. Malware-dna: Machine learning for malware analysis that treats malware as mutations in the software genome, 2023. Presented at the

Malware Technical Exchange Meeting (MTEM), Lawrence Livermore National Laboratory, California. July 25-27, 2023.

[67] M. E. Eren, N. Solovyev, R. Barron, M. Bhattarai, I.D. Boureima, K.O. Rasmussen, and B. S. Alexandrov. Sub-topic and semantic sub-structure extraction via split: Joint nonnegative matrix factorization (nmf) with automatic model selection. In *Conference on Data Analysis 2023 (CoDA 23')*, 2023.

[68] M. E. Eren, K. Rasmussen, B. Alexandrov, and C. Nicholas. Tensor decomposition methods for cyber security, 2024. Presented at the 15th Annual Malware Technical Exchange Meeting, McLean, Virginia, U.S. 2024.

[69] Maksim Eren, Nick Solovyev, Ryan Barron, Manish Bhattarai, Duc Truong, Ismael Boureima, Erik Skau, Kim Rasmussen, and Boian Alexandrov. Tensor Extraction of Latent Features (T-ELF), October 2023. URL `https://github.com/lanl/T-ELF`.

[70] Maksim Eren, Alexandrov Boian, and Charles Nicholas, editors. *Classifying Malware Using Tensor Decomposition*. Springer Nature, 2024.

[71] Maksim E. Eren, Manish Bhattarai, Nicholas Solovyev, Luke E.

Richards, Roberto Yus, Charles Nicholas, and Boian S. Alexandrov. One-shot federated group collaborative filtering. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 647–652, 2022. doi: 10.1109/ICMLA55696.2022.00107.

[72] Maksim E. Eren, Juston S. Moore, Erik Skau, Elisabeth Moore, Manish Bhattarai, Gopinath Chennupati, and Boian S. Alexandrov. General-purpose unsupervised cyber anomaly detection via non-negative tensor factorization. *Digital Threats*, feb 2022. ISSN 2692-1626. doi: 10.1145/3519602. URL `https://doi.org/10.1145/3519602`. Just Accepted.

[73] Maksim E. Eren, Nick Solovyev, Manish Bhattarai, Kim Ø. Rasmussen, Charles Nicholas, and Boian S. Alexandrov. Senmfk-split: Large corpora topic modeling by semantic non-negative matrix factorization with automatic model selection. In *Proceedings of the 22nd ACM Symposium on Document Engineering*, DocEng '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450395441. doi: 10.1145/3558100.3563844. URL `https://doi.org/10.1145/3558100.3563844`.

[74] Maksim E. Eren, Manish Bhattarai, Robert J. Joyce, Edward Raff, Charles Nicholas, and Boian S. Alexandrov. Semi-supervised classification of malware families under extreme class imbalance via hierarchical non-negative matrix factorization with automatic model selection. *ACM Trans. Priv. Secur.*, sep 2023. ISSN 2471-2566. doi: 10.1145/3624567. URL `https://doi.org/10.1145/3624567`. Just Accepted.

[75] Maksim E. Eren, Manish Bhattarai, Kim Rasmussen, Boian S. Alexandrov, and Charles Nicholas. Malwaredna: Simultaneous classification of malware, malware families, and novel malware. In *2023 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2023. doi: 10.1109/ISI58743.2023.10297217.

[76] Maksim E. Eren, Ryan Barron, Manish Bhattarai, Selma Wanna, Nicholas Solovyev, Kim Rasmussen, Boian S. Alcxandrov, and Charles Nicholas. Catch'em all: Classification of rare, prominent, and novel malware families. In *2024 12th International Symposium on Digital Forensics and Security (ISDFS)*, pages 1–6, 2024. doi: 10.1109/ISDFS60797.2024.10527250.

[77] Maksim Ekin Eren, Nick Solovyev, Chris Hamer, Renee McDonald, Boian Alexandrov, and Charles Nicholas. Covid-19 multidimensional kaggle literature organization. In *Proceedings of the ACM Symposium on Document Engineering 2021*, DocEng '21, New York, NY, USA, 2021. Association for Computing Machinery. doi: 10.1145/3469096.3474927. URL `https://doi.org/10.1145/3469096.3474927`.

[78] External Data Source. Virusshare dataset, 2018. URL `https://www.impactcybertrust.org/dataset%5Fview?idDataset=1271`.

[79] James Ezick, Tom Henretty, Muthu Baskaran, Richard Lethin, John Feo, Tai-Ching Tuan, Christopher Coley, Leslie Leonard, Rajeev Agrawal, Ben Parsons, and William Glodek. Combining tensor decompositions and graph analytics to provide cyber situational awareness at hpc scale. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2019. doi: 10.1109/HPEC.2019.8916559.

[80] Ming Fan, Jun Liu, Xiapu Luo, Kai Chen, Zhenzhou Tian, Qinghua Zheng, and Ting Liu. Android malware familial classification and representative sample selection via frequent subgraph analysis. *IEEE Trans-*

actions on Information Forensics and Security, 13(8):1890–1905, 2018. doi: 10.1109/TIFS.2018.2806891.

[81] Hadi Fanaee-T, Márcia D. B. Oliveira, João Gama, Simon Malinowski, and Ricardo Morla. Event and anomaly detection using tucker3 decomposition. *ArXiv*, abs/1406.3266, 2014.

[82] Cédric Févotte and A Taylan Cemgil. Nonnegative matrix factorizations as probabilistic inference in composite models. In *2009 17th European Signal Processing Conference*, pages 1913–1917. IEEE, 2009.

[83] Fabian H. Fonseca A, Serena Ferracci, Federico Palmaro, Luca Iocchi, Daniele Nardi, and Luisa Franchina. Static analysis of pe files using neural network techniques for a pocket tool. In *2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pages 01–06, 2021. doi: 10.1109/ICECCME52200.2021.9590958.

[84] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, 1975. doi: 10.1109/TIT.1975.1055330.

[85] Nicolas Gillis, Da Kuang, and Haesun Park. Hierarchical clustering of hyperspectral images using rank-two nonnegative matrix factorization. *IEEE Transactions on Geoscience and Remote Sensing*, 53(4):2066–2078, 2014.

[86] Derek Greene, Derek O'Callaghan, and Pádraig Cunningham. How many topics? stability analysis for topic models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 498–513. Springer, 2014.

[87] Rachel Grotheer, Yihuan Huang, Pengyu Li, Elizaveta Rebrova, Deanna Needell, Longxiu Huang, Alona Kryshchenko, Xia Li, Kyung Ha, and Oleksandr Kryshchenko. Covid-19 literature topic-based search via hierarchical nmf. *arXiv preprint arXiv:2009.09074*, 2020.

[88] Deepak Gupta and Rinkle Rani. Improving malware detection using big data and ensemble learning. *Computers & Electrical Engineering*, 86:106729, 2020.

[89] Steven Strandlund Hansen, Thor Mark Tampus Larsen, Matija Stevanovic, and Jens Myrup Pedersen. An approach for detection and family classification of malware based on behavioral analysis. In *2016*

International Conference on Computing, Networking and Communications (ICNC), pages 1–5, 2016. doi: 10.1109/ICCNC.2016.7440587.

[90] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[91] Winston Haynes. *Wilcoxon Rank Sum Test*, pages 2354–2355. Springer New York, New York, NY, 2013. ISBN 978-1-4419-9863-7. doi: 10.1007/978-1-4419-9863-7_1185. URL `https://doi.org/10.1007/978-1-4419-9863-7%5F1185`.

[92] David Hong, Tamara G. Kolda, and Jed A. Duersch. Generalized canonical polyadic tensor decomposition. *ArXiv*, abs/1808.07452, 2020.

[93] Jieqiong Hou, Minhui Xue, and Haifeng Qian. Unleash the power for tensor: A hybrid malware detection system using ensemble classifiers. In *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, pages 1130–1137, 2017. doi: 10.1109/ISPA/IUCC.2017.00170.

[94] Wenyi Huang and Jay Stokes. Mtnet: A multi-task neural network for dynamic malware classification. In *Proceedings of*

13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2016), pages 399–418. Springer, July 2016. ISBN 978-3-319-40667-1. URL `https://www.microsoft.com/en-us/research/publication/mtnet-multi-task-neural-network-dynamic-malware-classification/`.

[95] IBM. Cost of a data breach report. Technical report, IBM, 2021. URL `https://www.ibm.com/security/data-breach`.

[96] Paul Irofti and Andra Băltoiu. Malware identification with dictionary learning. In *2019 27th European Signal Processing Conference (EUSIPCO)*, pages 1–5, 2019. doi: 10.23919/EUSIPCO.2019.8903043.

[97] SM Ashiqul Islam, Yang Wu, Marcos Díaz-Gay, Erik N Bergstrom, Yudou He, Mark Barnes, Mike Vella, Jingwei Wang, Jon W Teague, Peter Clapham, et al. Uncovering novel mutational signatures by de novo extraction with sigprofilerextractor. *BioRxiv*, pages 2020–12, 2021.

[98] SM Ashiqul Islam, Marcos Díaz-Gay, Yang Wu, Mark Barnes, Raviteja Vangara, Erik N Bergstrom, Yudou He, Mike Vella, Jingwei Wang, Jon W Teague, et al. Uncovering novel mutational signatures by de

novo extraction with sigprofilerextractor. *Cell Genomics*, page 100179, 2022.

[99] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, oct 2002. ISSN 1046-8188. doi: 10.1145/582415.582418. URL https://doi.org/10.1145/582415.582418.

[100] Jianguo Jiang, Song Li, Min Yu, Gang Li, Chao Liu, Kai Chen, Hui Liu, and Weiqing Huang. Android malware family classification based on sensitive opcode sequence. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–7. IEEE, 2019.

[101] Xin Jin and Jiawei Han. *Mean Shift*, pages 806–808. Springer US, Boston, MA, 2017. ISBN 978-1-4899-7687-1. doi: 10.1007/978-1-4899-7687-1_532. URL https://doi.org/10.1007/978-1-4899-7687-1%5F532.

[102] Joblib Development Team. Joblib: running python functions as pipeline jobs, 2020. URL https://joblib.readthedocs.io/.

[103] Robert J. Joyce, Dev Amlani, Charles Nicholas, and Edward Raff.

Motif: A large malware reference dataset with ground truth family labels, 2021.

[104] Robert J. Joyce, Edward Raff, Charles Nicholas, and James Holt. Maldict: Benchmark datasets on malware behaviors, platforms, exploitation, and packers, 2023.

[105] Hideaki Kanehara, Yuma Murakami, Jumpei Shimamura, Takeshi Takahashi, Daisuke Inoue, and Noboru Murata. Real-time botnet detection using nonnegative tucker decomposition. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, page 1337–1344, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359337. doi: 10.1145/3297280.3297415. URL `https://doi.org/10.1145/3297280.3297415`.

[106] Kaspersky. Machine learning methods for malware detection. Technical report, 2020.

[107] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page

3149–3157, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[108] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

[109] Ilia Kisil, Ahmad Moniri, and Danilo P. Mandic. Tensor ensemble learning for multidimensional data. In *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1358–1362, 2018. doi: 10.1109/GlobalSIP.2018.8646694.

[110] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter development team. Jupyter notebooks - a publishing format for reproducible computational workflows. In Fernando Loizides and Birgit Scmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and*

Agendas, pages 87–90, Netherlands, 2016. IOS Press. URL `https://eprints.soton.ac.uk/403913/`.

[111] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, September 2009. doi: 10.1137/07070111X.

[112] Deguang Kong and Guanhua Yan. Discriminant malware distance learning on structural information for automated malware classification. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1357–1365, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450321747. doi: 10.1145/2487575.2488219.

[113] Danai Koutra, Evangelos E. Papalexakis, and Christos Faloutsos. Tensorsplat: Spotting latent anomalies in time. In *2012 16th Panhellenic Conference on Informatics*, pages 144–149, 2012. doi: 10.1109/PCi.2012.60.

[114] Da Kuang and Haesun Park. Fast rank-2 nonnegative matrix factorization for hierarchical document clustering. In *Proceedings of the 19th*

ACM SIGKDD international conference on Knowledge discovery and data mining, pages 739–747, 2013.

[115] Rajesh Kumar and S Geetha. Malware classification using xgboost-gradient boosted decision tree. *Advances in Science, Technology and Engineering Systems Journal*, 5(5):536–549, 2020.

[116] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[117] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18 (185):1–52, 2018. URL `http://jmlr.org/papers/v18/16-558.html`.

[118] Yeong Tyng Ling, Nor Fazlida Mohd Sani, Mohd Taufik Abdullah, and Nor Asilah Wati Abdul Hamid. Nonnegative matrix factorization and metamorphic malware detection. *Journal of Computer Virology and Hacking Techniques*, 15(3):195–208, 2019.

[119] Liu Liu, Bao-sheng Wang, Bo Yu, and Qiu-xi Zhong. Automatic malware classification and new malware detection using machine learning.

Frontiers of Information Technology & Electronic Engineering, 18(9): 1336–1347, 2017.

[120] Xinbo Liu, Yaping Lin, He Li, and Jiliang Zhang. A novel method for malware detection on ml-based visualization technique. *Computers & Security*, 89:101682, 2020. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2019.101682. URL `https://www.sciencedirect.com/science/article/pii/S0167404818314627`.

[121] Nicola Loi, Claudio Borile, and Daniele Ucci. Towards an automated pipeline for detecting and classifying malware through machine learning. *arXiv preprint arXiv:2106.05625*, 2021.

[122] Gabriel Maciá-Fernández, José Camacho, Roberto Magán-Carrión, Pedro García-Teodoro, and Roberto Therón. Ugr'16: A new dataset for the evaluation of cyclostationarity-based network idss. *Comput. Secur.*, 73:411–424, 2018. doi: 10.1016/j.cose.2017.11.004. URL `https://doi.org/10.1016/j.cose.2017.11.004`.

[123] David JC MacKay et al. Bayesian nonlinear modeling for the prediction competition. *ASHRAE transactions*, 100(2):1053–1062, 1994.

[124] Benjamin Marais, Tony Quertier, and Christophe Chesneau. Malware

analysis with artificial intelligence and a particular attention on results interpretability. In *Distributed Computing and Artificial Intelligence, Volume 1: 18th International Conference 18*, pages 43–55. Springer, 2022.

[125] Koji Maruhashi, Fan Guo, and Christos Faloutsos. Multiaspect-forensics: Pattern mining on large-scale heterogeneous networks with tensor analysis. In *2011 International Conference on Advances in Social Networks Analysis and Mining*, pages 203–210, 2011. doi: 10.1109/ASONAM.2011.80.

[126] Microsoft 365 Defender Threat Intelligence Team. Microsoft researchers work with intel labs to explore new deep learning approaches for malware classification, 2020. https://www.microsoft.com/security/blog.

[127] Aziz Mohaisen, Omar Alrawi, and Manar Mohaisen. Amal: High-fidelity, behavior-based automated malware analysis and classification. *Computers & Security*, 52:251–266, 2015. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2015.04.001. URL `https://www.sciencedirect.com/science/article/pii/S0167404815000425`.

[128] Morten Mørup and Lars Kai Hansen. Tuning pruning in sparse non-

negative matrix factorization. In *2009 17th European Signal Processing Conference*, pages 1923–1927. IEEE, 2009.

[129] Alexander B. Most, Maksim E. Eren, Boian S. Alexandrov, and Nigel Lawrence. Electrical grid anomaly detection via tensor decomposition. In *MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM)*, pages 162–169, 2023. doi: 10.1109/MILCOM58377.2023.10356348.

[130] Barath Narayanan Narayanan and Venkata Salini Priyamvada Davuluru. Ensemble malware classification system using deep neural networks. *Electronics*, 9(5), 2020. ISSN 2079-9292. doi: 10.3390/electronics9050721. URL `https://www.mdpi.com/2079-9292/9/5/721`.

[131] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. Malware images: Visualization and automatic classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, VizSec '11, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306799. doi: 10.1145/2016904.2016908. URL `https://doi.org/10.1145/2016904.2016908`.

[132] Benjamin T Nebgen, Raviteja Vangara, Miguel A Hombrados-Herrera, Svetlana Kuksova, and Boian S Alexandrov. A neural network for determination of latent dimensionality in non-negative matrix factorization. *Machine Learning: Science and Technology*, 2(2):025012, 2021.

[133] Andre T Nguyen, Edward Raff, Charles Nicholas, and James Holt. Leveraging uncertainty for improved static malware detection under extreme false positive constraints. *arXiv preprint arXiv:2108.04081*, 2021.

[134] Nick Njegomir. Using ai to develop enhanced cybersecurity measures. *Los Alamos National Laboratory*. URL `https://discover.lanl.gov/news/0215-ai-cybersecurity-measures/`.

[135] Rajvardhan Oak, Min Du, David Yan, Harshvardhan Takawale, and Idan Amit. Malware detection on highly imbalanced data through sequence modeling. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, AISec'19, page 37–48, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368339. doi: 10.1145/3338501.3357374.

[136] Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Criss-

man Loomis. Cupy: A numpy-compatible library for nvidia gpu calculations. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017. URL `http://learningsys.org/nips17/assets/papers/paper_16.pdf`.

[137] The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL `https://doi.org/10.5281/zenodo.3509134`.

[138] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library`.

pdf.

[139] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[140] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct): 2825–2830, 2011.

[141] Huu-Danh Pham, Tuan Dinh Le, and Thanh Nguyen Vu. Static pe malware detection using gradient boosting decision trees algorithm. In Tran Khanh Dang, Josef Küng, Roland Wagner, Nam Thoai, and Makoto Takizawa, editors, *Future Data and Security Engineering*, pages 228–236, Cham, 2018. Springer International Publishing. ISBN 978-3-030-03192-3.

[142] Panpan Qi, Wei Wang, Lei Zhu, and See Kiong Ng. *Unsupervised*

Domain Adaptation for Static Malware Detection Based on Gradient Boosting Trees, pages 1457–1466. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450384469. URL `https://doi.org/10.1145/3459637.3482400`.

[143] Yang Qi, Pierre Comon, and Lek-Heng Lim. Semialgebraic geometry of nonnegative tensor rank. *SIAM J. Matrix Anal. Appl.*, 37:1556–1580, 2016.

[144] Bernardo Quintero. Virustotal += bitdefender theta, 2019. URL `https://blog.virustotal.com/2019/10/virustotal-bitdefender-theta.html`.

[145] Bernardo Quintero. Virustotal += sangfor engine zero, 2019. URL `https://blog.virustotal.com/2019/11/virustotal-sangfor-engine-zero.html`.

[146] Vinayakumar R. and Soman K.P. Deepmalnet: Evaluating shallow and deep networks for static pe malware detection. *ICT Express*, 4(4):255–258, 2018. ISSN 2405-9595. doi: https://doi.org/10.1016/j.icte.2018.10.006. URL `https://www.sciencedirect.com/science/article/pii/S2405959518304636`.

[147] Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. Introduction to tensor decompositions and their applications in machine learning. *ArXiv*, abs/1711.10781, 2017.

[148] Edward Raff and C. Nicholas. A survey of machine learning methods and challenges for windows malware classification. *ArXiv*, abs/2006.09271, 2020.

[149] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K. Nicholas. Malware detection by eating a whole exe. In *AAAI Workshops*, 2018.

[150] Edward Raff, Charles K. Nicholas, and Mark McLean. A new burrows wheeler transform markov distance. In *AAAI*, 2020.

[151] Fauzan Hikmah Ramadhan, Vera Suryani, and Satria Mandala. Analysis study of malware classification portable executable using hybrid machine learning. In *2021 International Conference on Intelligent Cybernetics Technology Applications (ICICyTA)*, pages 86–91, 2021. doi: 10.1109/ICICyTA53712.2021.9689130.

[152] Hope Reese and Nick Heath. Inside amazon's clickworker platform:

How half a million people are being paid pennies to train ai. *TechRepublic. com*, 2016.

[153] S. Abijah Roseline, A. D. Sasisri, S. Geetha, and C. Balasubramanian. Towards efficient malware detection and classification using multilayered random forest ensemble technique. In *2019 International Carnahan Conference on Security Technology (ICCST)*, pages 1–6, 2019. doi: 10.1109/CCST.2019.8888406.

[154] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(87)90125-7. URL `https://www.sciencedirect.com/science/article/pii/0377042787901257`.

[155] Zakaria Sawadogo, Gervais Mendy, Jean Marie Dembele, and Samuel Ouya. Android malware detection: Investigating the impact of imbalanced data-sets on the performance of machine learning models. In *2022 24th International Conference on Advanced Communication Technology (ICACT)*, pages 435–441. IEEE, 2022.

[156] Tian Shi, Kyeongpil Kang, Jaegul Choo, and Chandan K Reddy. Short-

text topic modeling via non-negative matrix factorization enriched with local word-context correlations. In *Proceedings of the 2018 World Wide Web Conference*, pages 1105–1114, 2018.

[157] Michael Sikorski and Andrew Honig. *Practical malware analysis: the hands-on guide to dissecting malicious software*. no starch press, 2012.

[158] Nicholas Solovyev, Ryan Barron, Manish Bhattarai, Maksim E. Eren, Kim Ø. Rasmussen, and Boian S. Alexandrov. Interactive distillation of large single-topic corpora of scientific papers. In *2023 International Conference on Machine Learning and Applications (ICMLA)*, pages 1000–1005, 2023. doi: 10.1109/ICMLA58977.2023.00148.

[159] Sphinx Development Team. Sphinx 7.3.7+ documentation, May 2024. URL `https://www.sphinx-doc.org/en/master/`.

[160] Bowen Sun, Qi Li, Yanhui Guo, Qiaokun Wen, Xiaoxi Lin, and Wenhan Liu. Malware family classification method based on static feature extraction. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pages 507–513, 2017. doi: 10.1109/CompComm.2017.8322598.

[161] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *KDD '06*, 2006.

[162] Vincent YF Tan and Cédric Févotte. Automatic relevance determination in nonnegative matrix factorization with the/spl beta/-divergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35 (7):1592–1605, 2012.

[163] Engkarat Techapanurak, Masanori Suganuma, and Takayuki Okatani. Hyperparameter-free out-of-distribution detection using cosine similarity. In *Proceedings of the Asian conference on computer vision*, 2020.

[164] The Independent IT Security Institute. Malware statistics & trends report: Av-test, Oct 2021. URL `https://www.av-test.org/en/statistics/malware/`.

[165] The Independent IT Security Institute. Malware statistics & trends report: Av-test, Feb 2022. URL `https://www.av-test.org/en/statistics/malware/`.

[166] George Trigeorgis, Konstantinos Bousmalis, Stefanos Zafeiriou, and Bjoern Schuller. A deep semi-nmf model for learning hidden repre-

sentations. In *International Conference on Machine Learning*, pages 1692–1700. PMLR, 2014.

[167] Melissa JM Turcotte, Alexander D Kent, and Curtis Hash. Unified host and network data set. In *Data science for cyber-security*, pages 1–22. World Scientific, 2019.

[168] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL `http://www.jmlr.org/papers/v9/vandermaaten08a.html`.

[169] R. Vangara, E. Skau, G. Chennupati, H. Djidjev, T. Tierney, J. P. Smith, M. Bhattarai, V. G. Stanev, and B. S. Alexandrov. Semantic nonnegative matrix factorization with automatic model determination for topic modeling. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 328–335, 2020. doi: 10.1109/ICMLA51294.2020.00060.

[170] Raviteja Vangara, Erik Skau, Gopinath Chennupati, Hristo Djidjev, Thomas Tierney, James P Smith, Manish Bhattarai, Valentin G Stanev, and Boian S Alexandrov. Semantic nonnegative matrix factorization with automatic model determination for topic modeling. In

2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA), pages 328–335. IEEE, 2020.

[171] Raviteja Vangara, Manish Bhattarai, Erik Skau, Gopinath Chennupati, Hristo Djidjev, Thomas Tierney, James P Smith, Valentin G Stanev, and Boian S Alexandrov. Finding the number of latent topics with semantic non-negative matrix factorization. *IEEE Access*, 2021.

[172] R. Vinayakumar, Mamoun Alazab, K. P. Soman, Prabaharan Poornachandran, and Sitalakshmi Venkatraman. Robust intelligent malware detection using deep learning. *IEEE Access*, 7:46717–46738, 2019. doi: 10.1109/ACCESS.2019.2906934.

[173] Shuwei Wang, Qiuyun Wang, Zhengwei Jiang, Xuren Wang, and Rongqi Jing. A weak coupling of semi-supervised learning with generative adversarial networks for malware classification. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 3775–3782, 2021. doi: 10.1109/ICPR48806.2021.9412832.

[174] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the*

9th Python in Science Conference, pages 56–61, 2010. doi: 10.25080/Majora-92bf1922-00a.

[175] Kuo-Lung Wu and Miin-Shen Yang. Mean shift-based clustering. *Pattern Recogn.*, 40(11):3035–3052, nov 2007. ISSN 0031-3203. doi: 10.1016/j.patcog.2007.02.006. URL `https://doi.org/10.1016/j.patcog.2007.02.006`.

[176] Kun Xie, Xiaocan Li, Xin Wang, Gaogang Xie, Jigang Wen, Jiannong Cao, and Dafang Zhang. Fast tensor factorization for accurate internet anomaly detection. *IEEE/ACM Transactions on Networking*, 25(6): 3794–3807, 2017. doi: 10.1109/TNET.2017.2761704.

[177] Wei Xu, Xin Liu, and Yihong Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 267–273, 2003.

[178] Jinpei Yan, Yong Qi, and Qifan Rao. Detecting malware with an ensemble method based on deep neural network. *Security and Communication Networks*, 2018, 2018.

[179] David Yarowsky. Unsupervised word sense disambiguation rivaling su-

pervised methods. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*, ACL '95, page 189–196, USA, 1995. Association for Computational Linguistics. doi: 10.3115/981658.981684. URL `https://doi.org/10.3115/981658.981684`.

[180] Yanfang Ye, Tao Li, Yong Chen, and Qingshan Jiang. Automatic malware categorization using cluster ensemble. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 95–104, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450300551. doi: 10.1145/1835804.1835820.

[181] Baoguo Yuan, Junfeng Wang, Dong Liu, Wen Guo, Peng Wu, and Xuhua Bao. Byte-level malware classification based on markov images and deep learning. *Computers & Security*, 92:101740, 2020. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2020.101740. URL `https://www.sciencedirect.com/science/article/pii/S0167404820300262`.

[182] Lijun Zhang, Zhengguang Chen, Miao Zheng, and Xiaofei He. Robust

non-negative matrix factorization. *Frontiers of Electrical and Electronic Engineering in China*, 6(2):192–200, 2011.

[183] Shao-Huai Zhang, Cheng-Chung Kuo, and Chu-Sing Yang. Static pe malware type classification using machine learning techniques. In *2019 International Conference on Intelligent Computing and its Emerging Applications (ICEA)*, pages 81–86, 2019. doi: 10.1109/ICEA.2019.8858297.

[184] Xu-Yao Zhang, Guo-Sen Xie, Xiuli Li, Tao Mei, and Cheng-Lin Liu. A survey on learning to reject. *Proceedings of the IEEE*, 111(2):185–215, 2023.

[185] Yanxin Zhang, Yulei Sui, Shirui Pan, Zheng Zheng, Baodi Ning, Ivor Tsang, and Wanlei Zhou. Familial clustering for weakly-labeled android malware using hybrid representation learning. *IEEE Transactions on Information Forensics and Security*, 15:3401–3414, 2020. doi: 10.1109/TIFS.2019.2947861.

[186] Yunan Zhang, Chenghao Rong, Qingjia Huang, Yang Wu, Zeming Yang, and Jianguo Jiang. Based on multi-features and clustering ensemble method for automatic malware categorization. In

2017 IEEE Trustcom/BigDataSE/ICESS, pages 73–82, 2017. doi: 10.1109/Trustcom/BigDataSE/ICESS.2017.222.